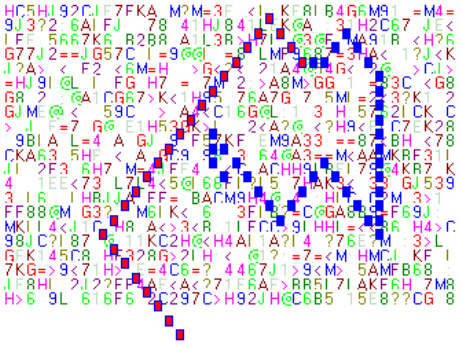
VidAs – Aufbau einer robusten, frei programmierbaren Maschine aus Assoziativmatrizen. Simulation und Hardware-Lösung.



Vom Fachbereich III
– Informations- und Kommunikationswissenschaften –
der Universität Hildesheim

zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

genehmigte

D i s s e r t a t i o n

von

Andreas Dierks
aus Hamburg

Berichterstatter:	Prof. Dr. Hans-Joachim Bentz Dipl.-Phys. Prof. Dr. Eberhard Schwarzer
eingereicht am:	29. 06. 2005
mündliche Prüfung am:	27. 09. 2005

Danksagung

Für die zahlreichen Anregungen auf dem Weg durch das Thema, die stets motivierende Ansprache und die geduldige Betreuung durch meinen Doktorvater, Herrn Professor Dr. Hans-Joachim Bentz, möchte ich mich ganz herzlich bedanken. Herrn Professor Dr. Eberhard Schwarzer danke ich für die bereitwillige Übernahme des Koreferats.

Überblick

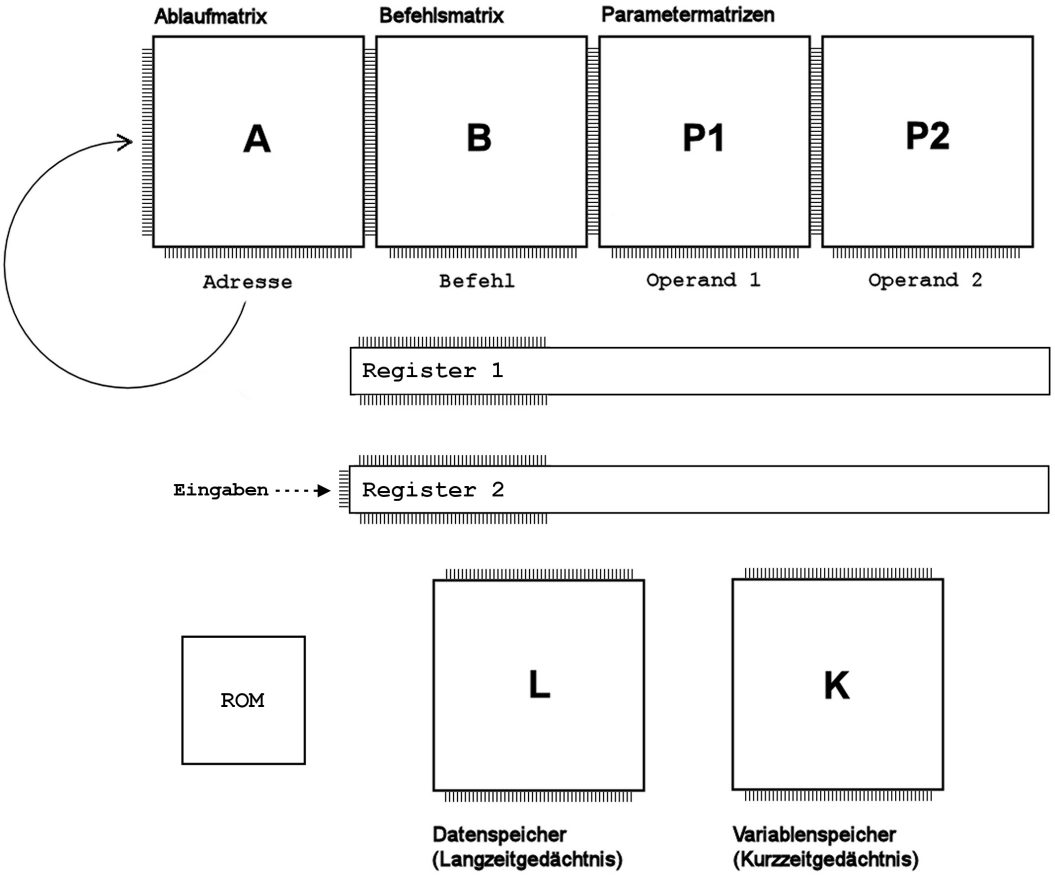


Abb. 1: Aufbau der VidAs -Maschine aus sechs Assoziativmatrizen (s. Kap. 4.9)

Erklärung

Hiermit versichere ich an Eides statt, dass ich diese Abhandlung selbständig und ohne unerlaubte Hilfe verfasst und die benutzten Hilfsmittel vollständig angegeben habe.

Hildesheim, den 28.06.2005

Kurzfassung

Durch Assoziativmatrizen gebildete Assoziativspeicher können als künstliche neuronale Netze aufgefasst werden, die sich zur fehlerrobusten Datenspeicherung, Mustererkennung, Musterergänzung und für zahlreiche daraus abgeleitete Aufgaben einsetzen lassen. Die vorliegende Arbeit weitet den Einsatzbereich von Assoziativmatrizen auf das störunanfällige Speichern und Abarbeiten von Programmen aus, indem mehrere Assoziativmatrizen zweier Typen zu einer frei programmierbaren Maschine zusammengesetzt werden. In der Programmierung dieser Maschine werden keine Zahlen oder Zähler eingesetzt, sondern Assoziationsketten benutzt. Das Anwendungsprogramm und seine Daten befinden sich gemeinsam im fehlertoleranten Assoziativspeicher, was verglichen mit anderen fehlertoleranten Systemen prinzipielle oder Geschwindigkeitsvorteile zeitigt.

Der Konstruktionsplan dieser Maschine (VIDAS -Maschine) wurde mit Hilfe eines Digitalsimulators detailliert erarbeitet. In der Maschine arbeiten sechs Assoziativmatrizen für verschiedene Aufgabenbereiche zusammen. Eine der Matrizen assoziiert eine Programmzeile mit der nächsten, eine weitere Matrix liefert die zu den Programmzeilen gehörenden Befehle und zwei weitere Matrizen übernehmen die Parameterversorgung. Variablen werden über eine besondere Assoziativmatrix mit schnellen Hebb-Synapsen verwaltet, damit Variablen ihre alten Werte „vergessen“ können. Daten werden hingegen in einer gewöhnlichen Assoziativmatrix abgelegt, um dem Anwendungsprogramm in der bewährten, fehlertoleranten Weise zur Verfügung zu stehen.

In der Programmierung der VIDAS -Maschine greift man bei der Konstruktion von Schleifen auf Assoziationsketten zu, da die Maschine kein Rechenwerk besitzt. Die Assoziative Programmierung erlaubt dennoch, der Maschine das Rechnen beizubringen.

Als Anwendungsfelder dieser neuen Technik werden solche vorgestellt, in denen Störungen oder Ungenauigkeiten herkömmliche Technik zu Fehlverhalten oder Ausfällen führen. Zum Test der Störunanfälligkeit wurde die VIDAS -Maschine in der Simulation mit vier Arten von Zerstörungen überprüft. Sie zeigte dabei ihre Störfestigkeit in der für den Einsatz von Assoziationmatrizen typischen Weise. Die Stärken der Maschine liegen zudem in der Fähigkeit des schnellen Lernens und Abfragens von Mustern, womit sie als Grundbaustein für den Aufbau von Systemen zur Mustererkennung dienen kann.

Abb. der Titelseite: Eine Figur hat einen Weg durch einen „Wald“ von Merkmalen gelernt und findet diesen nach einigem Herumirren letztlich über ihre Assoziativmatrizen wieder.

Abstract

Associative memory realized by associative matrices may be regarded as a special form of artifical neural networks which are well suited for error-tolerant information storage, pattern recognition and pattern completion and many other tasks derived from that. By this thesis it will be shown how to store and process programs in an error-tolerant manner by these matrices. Associative matrices of two types are assembled to a machine which will work even if some of its parts are disturbed or faulty. Programming this type of machine means not using numbers or counters but associative chains. The application and its data stay and work together in a bundle of fault-tolerant associative matrices. Compared to other fault-tolerant systems this produces advantages on principle or on time.

The VIDAS -machine was planned and constructed in detail by the help of a digital simulator. Six associative matrices are working together in the machine, each of them has its own task. One matrix is associating a line of the program with the following one, another matrix associates the command of the actual programline, two matrices are associating parameter values to supply the actual command with data. The values of variables are stored by a specific associative matrix which consists of fast Hebb-synapses because the former values of variables should be „forgotten“ as soon as possible. The data for the program is stored in a normal associative matrix. So, by accessing its data the program is able to use the fault-tolerant qualities of this kind of matrices.

Creating program loops for the VIDAS -machine is done by using associative chains because the machine doesn't „know“ numbers and has no arithmetic calculation unit (ALU). But by Associative Programming it is possible to teach the machine how to calculate.

Applications of this new kind of technics will be useful in situations when recent technics has problems because of inaccuracies or disturbances. Four kinds of disturbances were tested during the developement of the VIDAS -machine. As expected the fault-tolerance of the system was good and typical for the use of associative memory. Because of the ability of the machine to learn patterns very fast and to respond rapidly to a pattern, the machine will be useful as a basic module for the general pattern recognition task.

Versione ridotta

La memoria associativa costituita di matrici associative va interpretata come una forma speciale di rete neuronale che si può impiegare per memorizzare con tolleranza di errore delle informazioni, per riconoscere e completare dei modelli e per molte altre utilizzazioni a queste connesse e da queste derivate. Il presente lavoro estende il campo di applicazione delle matrici associative alla corretta – nonostante i disturbi – memorizzazione ed elaborazione di programmi nei quali più matrici associative di due tipi diversi vengono assemblate ad una macchina liberamente programmabile. Nella programmazione di questa macchina non vengono usati numeri o contatori, bensì catene associative. Il programma di applicazione e i suoi dati si trovano insieme nella memoria associativa a tolleranza di errore, presentando così dei vantaggi di principio o consentendo una maggior velocità rispetto agli altri sistemi a tolleranza di errore.

Questa macchina (VIDAS) è stata progettata e costruita dettagliatamente con l'aiuto di un simulatore digitale. Nella macchina sei matrici associative sono in opera insieme per diversi ambiti di applicazione: una delle matrici associa una linea del programma con la seguente, un'altra matrice fornisce gli ordini relativi alle linee del programma e altre due matrici si incaricano dell'alimentazione dei parametri. Le variabili vengono gestite da una matrice associativa particolare, con sinapsi Hebb veloci, in modo che le variabili possano „dimenticare“ il loro valore. I dati invece vengono depositati in una matrice associativa normale, tenuti a diposizione per l'utilizzo nel modo, già sperimentato con buoni risultati, della tolleranza all'errore.

Nella programmazione della macchina VIDAS si ricorre a catene associative, per costruire ripetizione, perchè la macchina non è dotata di calcolatore: tuttavia la programmazione associativa consente di insegnare alla macchina a calcolare.

Questa nuova tecnica può essere applicata e utilizzata nei casi in cui disturbi o inesattezze causino, nella tecnica tradizionale, un funzionamento difettoso o un guasto. Per testare il suo grado di resistenza ai disturbi la macchina VIDAS ha subito, in simulazione, quattro tipi diversi di interventi distruttivi; in questo esame la macchina ha dimostrato la sua stabilità e resistenza ai danni, alla maniera tipica delle matrici associative. Un altro punto di forza di questa macchina è la sua capacità di apprendere ed esaminare rapidamente dei modelli, ragion per cui può essere utilizzata come elemento di base nella costruzione di sistemi di riconoscimento di modelli.

Inhaltsverzeichnis

1 Einleitung

Eine angemessene Entgegnung auf die sich beim Betrieb von Rechnern einstellenden Fehler und Ausfälle bleibt in der Diskussion. Im Jahr 2001 setzte auf der internationalen Raumstation ISS ein Rechner aus und legte einen Roboterarm lahm.[1] Im Jahr danach versagte der Computer zum Umgang mit den Gyroskopen seinen Dienst, so dass die Raumstation sich nicht mehr korrekt zur Sonne hin ausrichten konnte.[2] Im Jahr 2004 brach in Frankreich für 20 Stunden das Netz des Mobilfunkanbieters Bouygues Telecom komplett zusammen, weil gleichzeitig zwei Server ausfielen.[3] Im August 2003 kam es im Nordosten der USA zum größten Blackout in der Geschichte, weil ein defekter Computer der Zentrale des Energieversorgers Überlastsituationen nicht mehr anzeigte.[4] Elektronikausfälle bei Kraftfahrzeugen haben 2003 um 23 Prozent zugenommen, wie die Gesellschaft für Informatik meldete. Ursachen vermutet man auf Seiten der Hardware in der Auswirkung elektromagnetischer Einstrahlung auf das Gerät, im Falle der Raumstation ISS in „unzureichend abgeschirmter kosmischer Strahlung“.[5]

Eine Kontrolle des Inhalts von speichernden Bauteilen liefert der Einsatz fehlererkennender und -korrigierender Codes.[6] Bei der Datenübertragung können ankommende Datenpakete, deren Prüfsumme auf einen Fehler hinweist, nochmals angefordert und nochmals übertragen werden. Gegen den Ausfall eines Rechners hilft der Parallelbetrieb mehrerer Rechner oder Laufwerke, so dass die Aufgaben eines ausfallenden Rechnerteils gegebenenfalls sogleich vom Parallelsystem übernommen werden können. Server dieser Art werden von den Herstellern zum Beispiel mit zwei oder vier Prozessoren („Vier-Wege-Server“) versehen[7] und als „besonders ausfallsicher“ oder „hoch verfügbar (fault-tolerant)“ bezeichnet. In der Chip-Herstellung setzt man auf Redundanz, um fehlerhafte Zellen zu ersetzen.[8]

Eine andere Herangehensweise für den Umgang mit Speicherfehlern liefern seit einiger Zeit Assoziativmatrizen, die sich hinsichtlich solcher Fehler „robust“ zeigen, das heißt, der Zugriff auf die Daten erfolgt in weiten Grenzen trotzdem erfolgreich, auch wenn Matrixteile zerstört werden. Das VidAs 3-Projekt, welches in Kapitel 4.6 vorgestellt wird, untersuchte das Verhalten eines Verbunds von Assoziativmatrizen hinsichtlich verschiedener, zufallsgesteuerter Zerstörungsarten.

[1] vgl. Heise Online vom 19.07.01
[2] vgl. Stuttgarter Zeitung vom 05.02.02
[3] vgl. Heise Online vom 18.11.04
[4] vgl. Heise Online vom 20.11.03
[5] s. Heise Online vom 19.07.01
[6] vgl. [Hamming 87, Kap. 2 und 3]
[7] vgl. Heise online vom 11.03.03
[8] vgl. Israel KOREN: „Fault Tolerant Computing“, S. 12, http://euler.ecs.umass.edu/ece655/pdf/Part16-dft-sh.pdf, Frühjahr 2005

Das Projekt VIDAS 4 prüfte die Robustheit von Assoziativmatrizen bei der Verwendung als Programmspeicher (s. Kapitel 4.7). Es stellte sich im Anschluss an die aufschlussreichen Tests die Frage, ob man an die Stelle des oben genannten aufwändigen Parallelbetriebs von Rechnern oder von Laufwerken nicht auch eine auf Assoziativmatrizen basierende Technik setzen kann. Ab Kapitel 4 wird eine solche Maschine vorgestellt. Sie kommt beim Bau ihrer Assoziativmatrizen ohne Zentralprozessoren aus, da sie fast ausschließlich eine vereinfachte HEBBsche Lernregel[9] nutzt. Die VIDAS -Maschine setzt sich aus einfachen Schaltgattern zusammen, die die Funktionsweise mehrerer, zusammenwirkender Assoziativmatrizen vollständig übernehmen. Es ergibt sich dabei eine gewisse Ähnlichkeit zur Wahrnehmung von unterschiedlichen Aufgaben durch unterschiedliche Regionen eines Gehirns. Die schaltungstechnischen Details liefert Kapitel 4.9. Anwendungsbeispiele für einige Modelle der VIDAS -Maschine zeigt Kapitel 6.

Da die VIDAS -Maschine frei programmierbar ist, stellt sie selbst einen Prozessor dar, mit dem sich ein Universalrechner aufbauen ließe. Die Programmierung dieser Maschine, hier *Assoziative Programmierung* genannt, nutzt als Paradigma die Assoziationskette, durch die Schleifenstrukturen gebildet werden. Einen Überblick zur Assoziativen Programmierung gibt Kapitel 5.

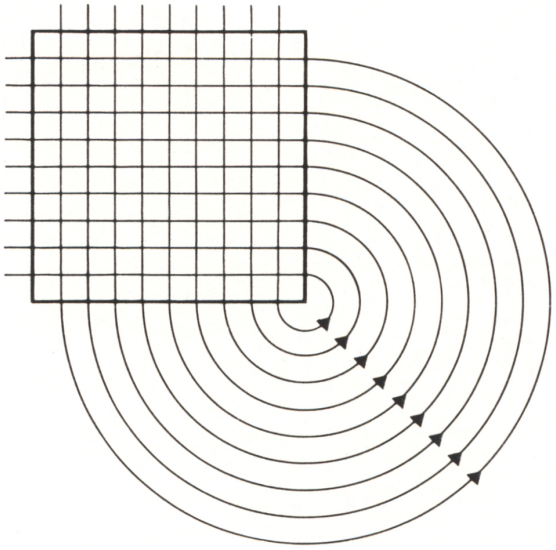


Abb. 2: Rückkopplung zur Mustervervollständigung bei einer Assoziativmatrix

Schon 1980 wies Günther PALM in „On Associative Memory“[10] darauf hin, dass sich Assoziativmatrizen sowohl zur Mustererkennung als auch zur Mustervervoll-

[9] vgl. [Palm 88, S. 58]
[10] vgl. [Palm 80]

ständigung einsetzen lassen. Dort stellte er in seinem Ansatz, Assoziativspeicher als Gehirnmodell aufzufassen, auch eine Rückkopplungstechnik vor, die zur Mustererkennung und -vervollständigung eingesetzt wurde (s. Abb. 2). PALM zeigt in „Assoziatives Gedächtnis und Gehirntheorie“[11], dass sich mittels Rückkopplung eine Sequenz in derselben Reihenfolge abrufen lässt, in der sie abgespeichert worden ist. Dies war Ausgangspunkt für den Ansatz des VIDAS -Projekts, das dort zum Abruf einer Folge von Mustern dienende Konzept aufzugreifen und ein Muster als Schritt eines Programms aufzufassen, welcher mit dem Nachfolgeschritt assoziiert wird. Beginnend mit Kapitel 2 sei im Folgenden nachvollzogen, was zum Aufbau einer „hoch verfügbaren“ Maschine aus Assoziativmatrizen führt.

Zur Begriffsklärung sei an dieser Stelle erwähnt, dass mit dem in der Literatur zu findenden Ausdruck „associative processor“ ein Bauteil bezeichnet wird, welches Daten parallel in einen inhaltsadressierbaren Speicher schreibt.[12] Für die in dieser Arbeit vorgestellte, frei programmierbare Maschine greift diese Bezeichnung daher deutlich zu kurz, hat mit dem Thema „fault-tolerant associative processors“ jedoch die Absicht gemein, als eine ihrer Aufgaben einen fehlertoleranten, schnellen Datenzugriff zu ermöglichen.

[11] vgl. [Palm 88, S. 60-61]

[12] „Associative memory concerns the concept that one idea may trigger the recall of a different but related idea. Traditional computers, however, rely upon a memory design that stores and retrieves data by its address rather than its content. In such a search, every accessed data word must travel individually between the processing unit and the memory. [...] Associative memory, in contrast, provides a naturally parallel and scalable form of data retrieval for both structured data (e.g. sets, arrays, tables, trees and graphs) and unstructured data (raw text and digitized signals). An associative memory can be easily extended to process the retrieved data in place, thus becoming an associative processor. This extension is merely the capability for writing a value in parallel into selected cells.“, aus: Krikelis, A. und Weems, C.C.: „Associative processing and processors“ in: *Computer*, November 1994, Vol. 27, Issue 11, S. 12-17

2 Nervenzellen und Assoziativmatrizen

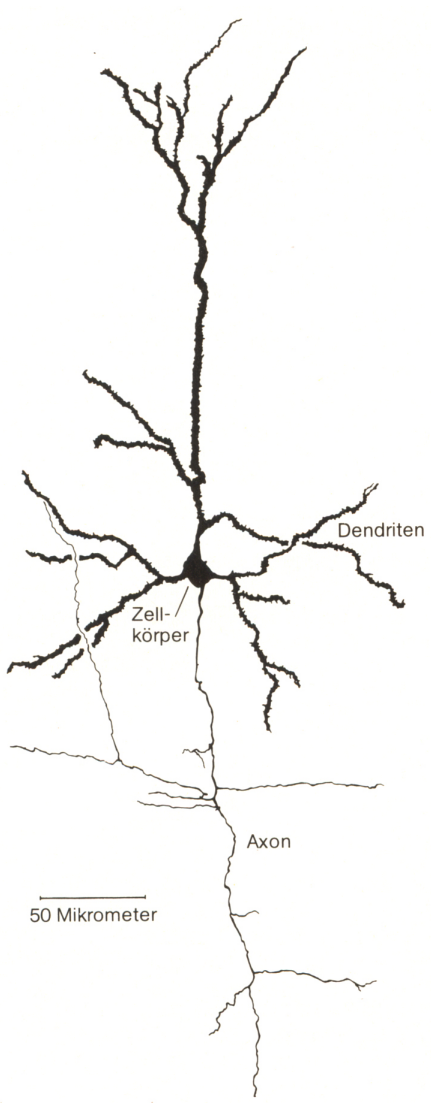


Abb. 3: Nervenzelle

Das Wissen um die Funktionsweise einer Nervenzelle (Neuron) dient als Vorlage für den Entwurf von Modellen, durch die man Leistungen des Gehirns nachzubilden versucht. Über ihre Dendriten nimmt die Nervenzelle die Signale anderer Zellen auf und generiert daraus ein Ausgabesignal, welches sie über ihr Axon weiterleitet. Die Eingabesignale erfahren eine Verstärkung oder Dämpfung bevor sie zum Ausgabesignal beitragen. Dieser Faktor wird im Folgenden *Gewicht* genannt. Das Ausgabesignal läuft in alle Zweige des Axons[13] und wird über synaptische Verbindungen von den Dendriten anderer Nervenzellen empfangen. Eine Nervenzelle besitzt etwa 10^4 Synapsen an ihren Dendriten. Im Mittel, so folgert PALM, müssten sich auf dem Axon einer Nervenzelle also ebenfalls 10^4 Synapsen befinden. Wegen der $2 \cdot 10^{10}$ Nervenzellen des menschlichen Gehirns, umfasse dieses somit etwa 10^{14} Synapsen.

Bei diesen Überlegungen bleiben die Beobachtungen unberücksichtigt, die R. Douglas Fields 2004 in „Die unbekannte Seite des Gehirns — Wie Gliazellen im Kopf mitreden“[14] zusammenfasst. Die Gliazellen[15] fügen dem Netzwerk aus Neuronen ein weiteres, umspannendes Netz hinzu, welches Signale im Unterschied zu ersterem nicht elektrisch sondern chemisch überträgt. Einige Arten von Gliazellen (Astrozyten, Oligodendrozyten, Schwannzellen) stellen über Botenstoffe Kontakte zum Netz der Nervenzellen her und nehmen auf dieses Einfluss. Das menschliche Gehirn enthält neunmal so viele Glia- wie Nervenzellen.[16] Diese neueren Erkennt-

[13] vgl. [Palm 82, S. 10]

[14] vgl. [Fields 04]

[15] glia — Leim, Kitt

[16] „Lange stand die so genannte Glia im Schatten der Neuronen, galt sie doch vor allem als Stützstruktur im Gehirn. Doch sie scheint vieles zu beeinflussen – sogar, wie das Gehirn denkt, lernt und sich erinnert.“, aus: [Fields 04, S. 46]

nisse geben Anlass, Wechselwirkungen zwischen verschieden arbeitenden Netzen im Gehirn anzunehmen und gegebenenfalls nachzubilden.

2.1 Grundlagen nach PALM

Im grundlegenden Werk „Neural Assemblies - An Alternative Approach to Artificial Intelligence“ stellt Günther PALM in dem Kapitel “How to build well-behaving machines“[17] das bereits 1943 beschriebene Schwellwert-Neuron von McCULLOCH und PITTS (s. Abb. 4) als Modell einer Nervenzelle vor.

Dieses Modell zeichnet das Ein- und Ausgabeverhalten einer Nervenzelle grob nach, indem die über die n Synapsen einlaufenden Signale x_i mit den Gewichtsfaktoren w_i multipliziert werden, bevor die Summe darüber entscheidet, mit welcher Ausgabe das Axon der Nervenzelle reagiert. Ist die Summe $\sum_{i=1}^{n} w_i x_i$ größer als ein Schwellenwert θ, liefert das Modell-Neuron eine „1“, sonst eine „0“ im Ausgang. Sowohl in den Eingängen x_i als auch im Ausgang dieses Modell-Neurons wird nur zwischen den beiden Zuständen „0“ und „1“ unterschieden. Diese Grundannahme gilt dann im Weiteren auch für Assoziativmatrizen.

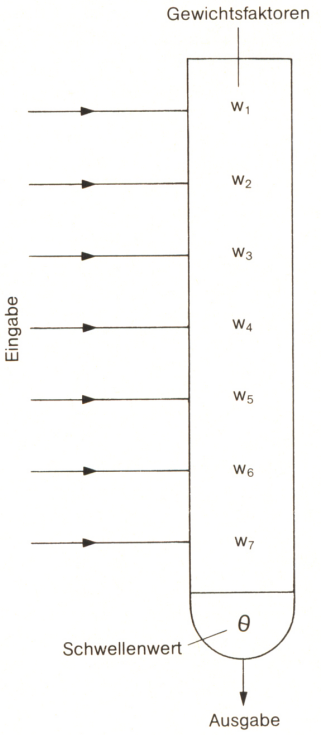


Abb. 4: Modell einer Nervenzelle

Dass und wie sich jede Eingabe für eine gedachte Maschine im Sinne seines Gehirnmodells durch eine Bitfolge darstellen lässt, klärt der Autor im gleichen Kapitel. Anschließend nutzt er die aussagenlogische Erkenntnis, dass jede aussagenlogische Funktion durch eine Verknüpfung von Konjunktion, Disjunktion und Negation dargestellt werden kann (zum Beispiel die Subjunktion $p \rightarrow q$ durch $\overline{p} \vee q$, mit p und q als Wahrheitswertevariablen), und sogar allein die NAND-Funktion ausreicht,[18] um jeden aussagenlogischen Ausdruck durch sie darzustellen. Damit weist er nach, dass das Schwellwert-Neuron von McCULLOCH und PITTS ebenfalls in der Lage ist, jede aussagenlogische Funktion darzustellen. Das Schwellwert-Neuron bildet die NAND-Funktion durch $w_1 = -1, w_2 = -1$ und den Schwellenwert $\theta = -1,5$ nach.

Aus einer Anreihung dieser Schwellwert-Neuronen entstehen Assoziativmatrizen wie in Abbildung 5.

[17] s. [Palm 82, Kap. 3]

[18] wg. $\overline{p} = p \barwedge p,\ p \wedge q = \overline{p \barwedge q},\ p \vee q = \overline{p} \barwedge \overline{q}$, mit $\barwedge$ als Bezeichner der NAND-Funktion

2.2 Lern- und Abfrageregel der Assoziativmatrix

Unter der Überschrift „The improved Matchbox Algorithm“[19] stellt PALM die Assoziativmatrix vor, die im VIDAS -Projekt (bis auf eine Ausnahme beim sogenannten „Kurzzeitgedächtnis“, s. Kap. 4.1, unten) eingesetzt wird. Er nutzt die Assoziativmatrix dort, um einer Situation auf dem Schachbrett einen Schachzug zuzuordnen, und verweist im Weiteren auf seine Ergebnisse von 1980, denen zufolge man in eine solche Assoziativmatrix mehrere (binäre) Frage-/Antworttupel eintragen und mit x Matrixpositionen etwa $\frac{2}{3} \cdot x$ Bit speichern kann (für große x).[20]

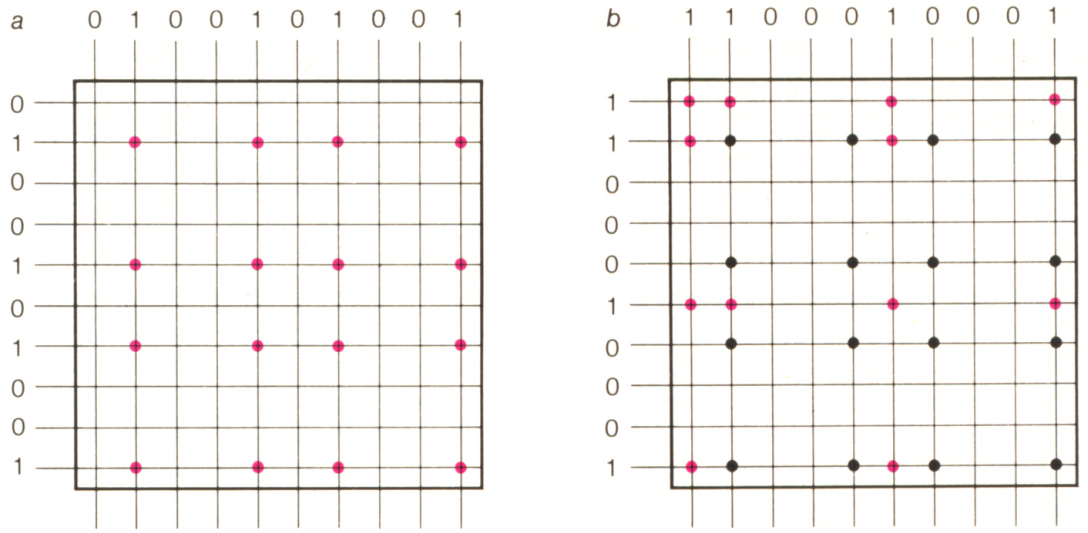


Abb. 5: Lernregel für Assoziativmatrizen

Als Lernregel für Assoziativmatrizen wird in „Parallel Processing for Associative and Neuronal Networks“ die Gleichung $A' = A + \mathfrak{x} \cdot \mathfrak{y}^t$ angegeben,[21] wobei A' die Matrix nach dem Lernen, A die Matrix vor dem Lernen, $\mathfrak{x}$ das Fragetupel und $\mathfrak{y}^t$ das transponierte Antworttupel bezeichnet. Als Matrizenaddition ist hier offenbar eine aussagenlogische ODER-Verknüpfung der Koeffizienten der Operandenmatrizen gemeint, wie PALMs Veranschaulichung[22] in Abb. 5 verdeutlicht: in a) wird ein erstes Frage-/Antworttupel eingetragen, in b) kommt ein zweites Paar hinzu.[23]

[19] vgl. [Palm 82, Kap. 5]
[20] Zum Ausnutzungsgrad von 69 % s. Kap. 8
[21] vgl. [Palm84, S. 201]
[22] aus [Palm 88, S. 58]
[23] Fragetupel an den horizontalen, Antworttupel an den vertikalen Axonen

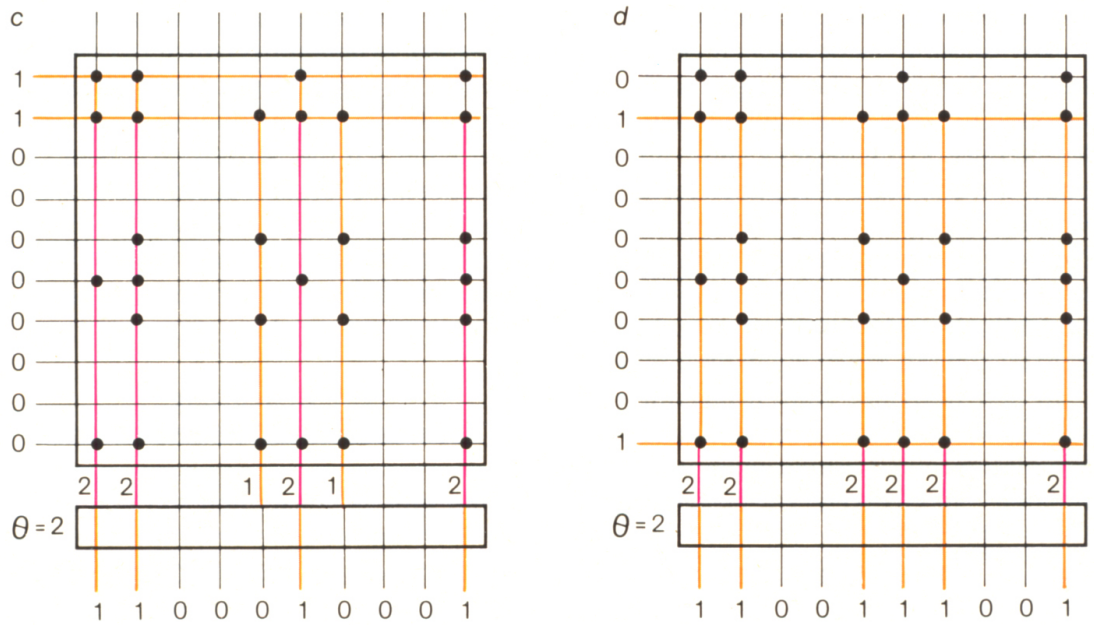


Abb. 6: Abfragen einer Assoziativmatrix

Als zweite grundlegende Matrixoperation gibt PALM die Abfrageregel durch $\mathfrak{y} = \mathfrak{x} \cdot A$ an,[24] der meist eine Schwellwertoperation $\mathfrak{z} = f_\theta(\mathfrak{y})$ folge, für die gilt:

$$z_i = \left\{ \begin{array}{ll} 1, & falls\ y_i \geq \theta \\ 0, & sonst \end{array} \right\}$$

Als Produkt von Fragetupel $\mathfrak{x}$ und Matrix A ist hier $\mathfrak{x}^t \cdot A$ gemeint, wie seine Veranschaulichung in Abb. 6 erklärt.

Im Abschnitt c) von Abb. 6 wird mit Teilen eines gelernten Fragetupel abgefragt und gezeigt, wie sich dennoch das zugehörige Antworttupel ergibt. Im Abschnitt d) wird mit einer Mischung aus Teilen zweier verschiedener Fragetupel abgefragt und verdeutlicht, dass das resultierende Antworttupel dann aus Teilen der zwei zugehörigen Antworttupel besteht, falls die beiden Fragetupelteile gleich viele Einsen in die Abfrage einbringen.

Für Teile der VIDAS -Maschine, die bewirken, dass ein Programmschritt mit dem nächsten assoziiert wird, müssen Fälle wie in Abb. 6 d) vermieden werden, damit sich bei der Programmausführung keine Endlosschleifen ergeben. Man erreicht das, indem man für alle Skalarpodukte von je zwei Matrixspalten A_i und A_j, mit $i \neq j$, $i, j = 1, ..., n$, des betreffenden Abschnitts der Assoziativmatrix A

[24] s. [Palm84, S. 201]

fordert: $A_i \cdot A_j = 0$ (Orthogonalitätsbedingung). Das führt zwar zu einer kleineren möglichen Anzahl an Programmschritten, fördert andererseits jedoch die Störunanfälligkeit.

Da das Abfragen der Assoziativmatrizen in der VIDAS -Maschine in den Matrixspalten parallel (synchron) erfolgt und zudem der Schwellenwert θ immer gleich dem Maximum $\overset{max}{j}(s_j)$ der Einträge im Schwellwerttupel $\mathfrak{s} = \mathfrak{x}^t \cdot A$ für $j = 1, ..., n$ ist, lässt sich hier die Abfrageregel wie folgt beschreiben: Seien $\mathfrak{x}$ das Fragetupel, $\mathfrak{s}$ das Schwellwerttupel, $\mathfrak{y}$ das Antworttupel, A die Assoziativmatrix und A_i die i. Spalte von A. Dann geschieht das Abfragen für alle $i = 1, ..., n$ durch $s_i = A_i \cdot \mathfrak{x}$ und

$$y_i = \left\{ \begin{array}{l} 1, \ falls \ s_i = \overset{max}{j}(s_j) \\ 0, \ sonst \end{array} \right\}$$

Um im Rahmen des Variablenkonzepts der VIDAS -Maschine zu ermöglichen, dass eine Variable, der durch eine Assoziativmatrix K ihr Variablenwert zugeordnet wird, diesen durch einen neuen ersetzt bekommen kann (und den vorherigen Wert also „vergisst"), wurde in der oben angegebenen Lernregel $A' = A + \mathfrak{x}^t \cdot \mathfrak{y}$ als Matrizenaddition die aussagenlogische UND-Verknüpfung gewählt.

2.3 Anwendungsgebiete für Assoziativmatrizen

Einen Zusammenhang zwischen Assoziativmatrizen und neuronalen Netzwerken stellt PALM im Artikel „Das Gehirn als assoziativer Speicher" her.[25] Er unterscheidet dabei zwischen modifizierbaren und nicht modifizierbaren synaptischen Verbindungen.

Einen Überblick gibt dazu Abb. 7. Die horizontalen Axone „bedienen" die modifizierbaren synaptischen Verbindungen, die sich auf den Dendriten befinden, und die vertikalen Axone „schalten" die Dendriten ein, damit die horizontalen Axone sich auswirken können. Dieses Prinzip findet sich zur Lernphase der in der VIDAS -Maschine eingesetzten Assoziativmatrizen wieder, indem vertikale Axone (Leitungen) als Schreib-Signalgeber für die Speicherelemente der Matrixspalte fungieren. Wenn von vertikalen und horizontalen Axonen gleichzeitig ein Signal bei den modifizierbaren Synapsen eintrifft, nähme deren „Übertragungsstärke" zu.[26] Im Falle der VIDAS -Maschine wird einer solchen synaptischen Verbindung dann der Wert „1" zugewiesen.

[25] vgl. [Palm 88]
[26] s. [Palm 88, S. 60]

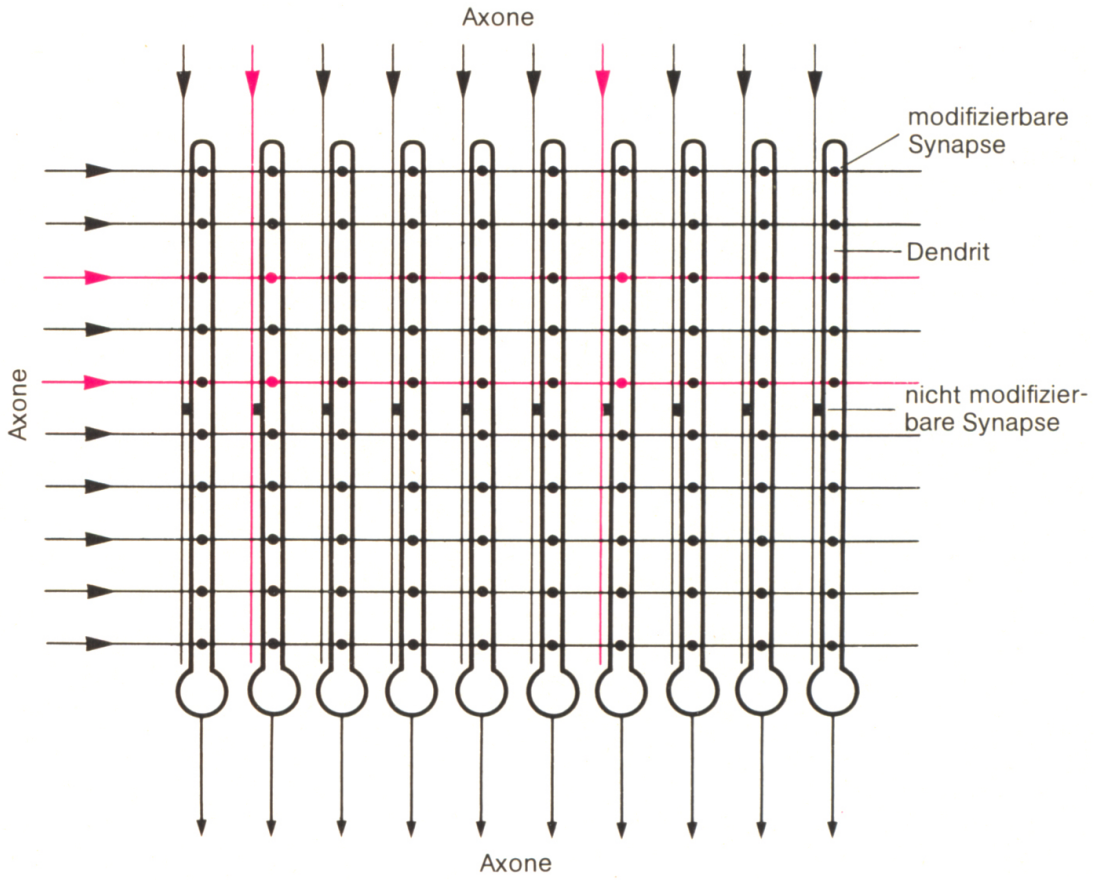


Abb. 7: Assoziativmatrix als neuronales Netz

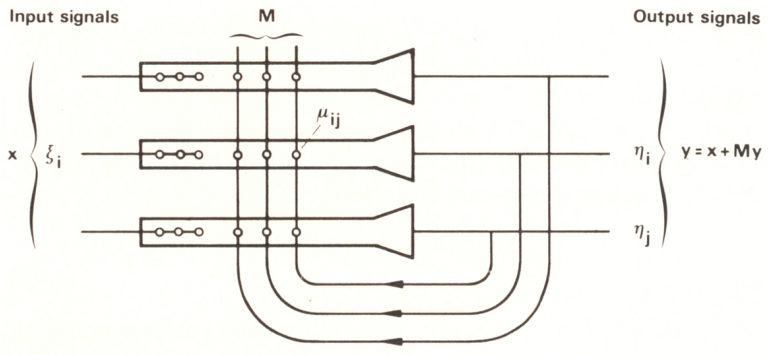


Abb. 8: Rückgekoppeltes neuronales Netzwerk nach KOHONEN

Als Anwendungsmöglichkeit einer solchen neuronalen Struktur wird die Mustervervollständigung genannt und dabei auf die Bedeutung der Rückkopplung ver-

wiesen.[27] Teuvo KOHONEN schreibt dazu in „Self-Organization and Associative Memory“ im Kapitel „Adaptive Feedback Networks“[28]: „In order to analyze the effect of feedback in a pure, isolated form, we first assume that every basic adaptive unit receives a single external input signal, and feedback signals from all the other units. [...] Without the feedback connections, the external signals would only be propagated unaltered to the output ports.“

Abb. 9: Mustervervollständigung mit einem Assoziativspeicher nach KOHONEN

Abb. 8 gibt diese Idee wieder. Die Eingangssignale ξ_i werden auf die Ausgangssignale η_i abgebildet, indem die durch die Gewichtsmatrix M dargestellte Rück-

[27] vgl. [Palm 88, S. 61-62]
[28] s. [Kohonen 88, S. 105]

kopplung zu allen ξ_i das Produkt $\mu_i \cdot \eta_i$ addiert. Fehlt die Rückkopplung, ist M gleich der Nullmatrix und folglich werden die Eingangssignale ξ_i unverändert auf die Ausgänge gelegt.

Die erfolgreiche Mustervervollständigung durch einen rückgekoppelten Assoziativspeicher zeigt Abb. 9. In der linken Spalte sind die vom Assoziativspeicher gelernten Muster abgebildet, in der Mitte finden sich oben ein verrauschtes und unten ein unvollständiges Muster, mit denen der Assoziativspeicher dann abgefragt wurde, und in der rechten Spalte erkennt man, wie gut er die gestörten Muster wieder ergänzen konnte.[29]

Eine Mustervervollständigung ist auch mit den oben vorgestellten Assoziativmatrizen möglich, wie PALM in [Palm 88, S. 62] am Beispiel von Abb. 10 erklärt. Das vollständige Muster ist durch Verdickungen an den synaptischen Verbindungen zwischen horizontalen und vertikalen Axonen kenntlich gemacht. Abgefragt wird die Matrix aber nur durch die Aktivierung von zwei vertikalen Axonen links. Dank der Rückkopplung kommt es im Nachwege auch zur Aktivierung der zwei vertikalen Axone rechts, so dass sich an den Ausgängen letztlich das gesamte gelernte Muster einstellt.

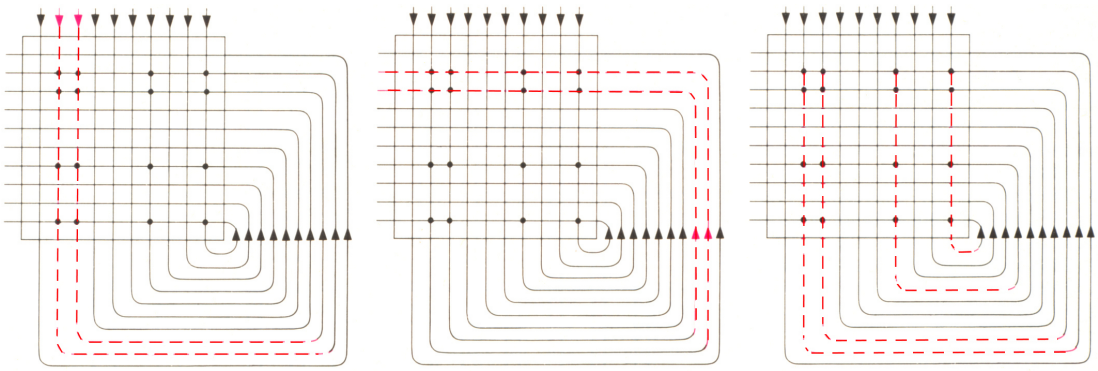


Abb. 10: Mustervervollständigung durch eine Assoziativmatrix

Außer zur Mustervervollständigung sieht PALM das Anwendungsgebiet von Assoziativmatrizen in Anwendungen, die auf die Fehlertoleranz der Matrizen setzen.[30]

[29] s. [Kohonen 88, S. 162] und [Palm 88, S. 55]

[30] PALM schreibt in „The PAN System and the WINA Project“ (s. [Palm 93]): „Associative memories are used for two types of tasks: fault tolerant pattern mapping and pattern completion. As a typical example, consider the search for some data by means of a key. If the key points directly to the data, this is called pattern mapping; if this also works for incomplete or distorted keys, the pattern mapping is called fault tolerant; if the key is itself a part of the data, we are dealing with pattern completion. Thus, pattern completion can be regarded as a special case of fault tolerant pattern matching, where the data are the keys.“

Eine Reihe weiterer Veröffentlichungen von PALM zielen auf die Erklärung der Leistung des Gehirns im Bereich des Klassifizierens, des Erkennens von Tönen und Bildern. Auch zur Lösung der Probleme bei der Verwirklichung autonomer, mobiler Roboter wird beigetragen.

Die Leistung des Gehirns beim Erinnern hebt Hans-Joachim BENTZ in „Ein Gehirn für den PC“[31] wie folgt hervor: „Wenn ich Ihnen die Frage stelle: »Wissen Sie die Höhe (im Vergleich zur Meereshöhe) des Amsterdamer Flughafens Schiphol?«, so werden Sie (mit großer Wahrscheinlichkeit) 'nein' sagen. Zu dieser Antwort sind Sie spontan fähig. Weder Sie noch ich haben den Eindruck, dass Sie Ihr ganzes Orts- und Zahlengedächtnis durchforsten müssen, um zur Antwort zu kommen. Der Zeitbedarf dafür ist gering und außerdem praktisch der gleiche, ob Sie's nun wissen oder nicht. [...] Hätte ich in meiner eben gestellten Frage 'Amschterdam' geschrieben, so hätten Sie mich vermutlich dennoch verstanden.“ Ausgehend von dieser Beschreibung schildert er die Vorteile der Assoziativmatrizen beim schnellen Finden gestörter oder unvollständiger Informationen und die Ähnlichkeit der Matrixleistung zur Leistung des menschlichen Gehirns. BENTZ setzt in den Folgejahren eine Reihe von Projekten zur Textrecherche, zur Zeichenerkennung, zur Datenanalyse, zur Ähnlichkeit von Texten, zur Algorithmik, zum Lernen und Lehren in die Tat um, Projekte, in denen die fehlertoleranten, ähnlichkeitserkennenden Eigenschaften der Assoziativmatrizen eine zentrale Rolle spielen. Das Augenmerk richtet sich in diesen Projekten meist auf die angemessene Kodierung der Frage- und Antworttupel bevor diese in die Assoziativmatrix eingetragen wird. Auch im VIDAS -Projekt spielt die geeignet gewählte Kodierung eine Rolle, damit Endlosschleifen vermieden werden, damit Assoziationsketten terminieren, damit Variablenwerte erhalten bleiben. Kapitel 4.11 und 8 bieten dazu nähere Erläuterungen.

[31] s. [Bentz 88, S. 86]

3 Hardwarelösungen für Neuronale Netze

Nervenzellen werden oft als eigenständig arbeitende Einheiten empfunden und als solche werden ihre Nachbildungen in künstliche neuronale Netzwerke eingesetzt, in denen sie parallel „rechnen“ könnten. Universalrechner, die die Arbeitsweise eines Netzes von Nervenzellen simulieren sollen, setzen das Nebeneinander der Nervenzellenaktivitäten gezwungenermaßen in ein Nacheinander um, da in der Regel nicht für jede Nervenzelle ein Prozessor bereit gestellt werden kann. In zeitkritischen Anwendungen wird folglich der Ruf nach Neurohardware laut, die den Parallelbetrieb der eigenständigen Einheiten unterstützt. Dieses gilt auch für den Umgang mit Assoziativmatrizen. Sowohl das Lernen als auch das Abfragen kann in den Matrixspalten parallel erfolgen, wie es die Formulierung der Abfrageregel in Kapitel 2.2 als nebenläufig zu erledigende Berechnung der Skalarprodukte zwischen Matrixspalten und Fragetupel nahelegt.

3.1 Lernmatrix nach STEINBUCH

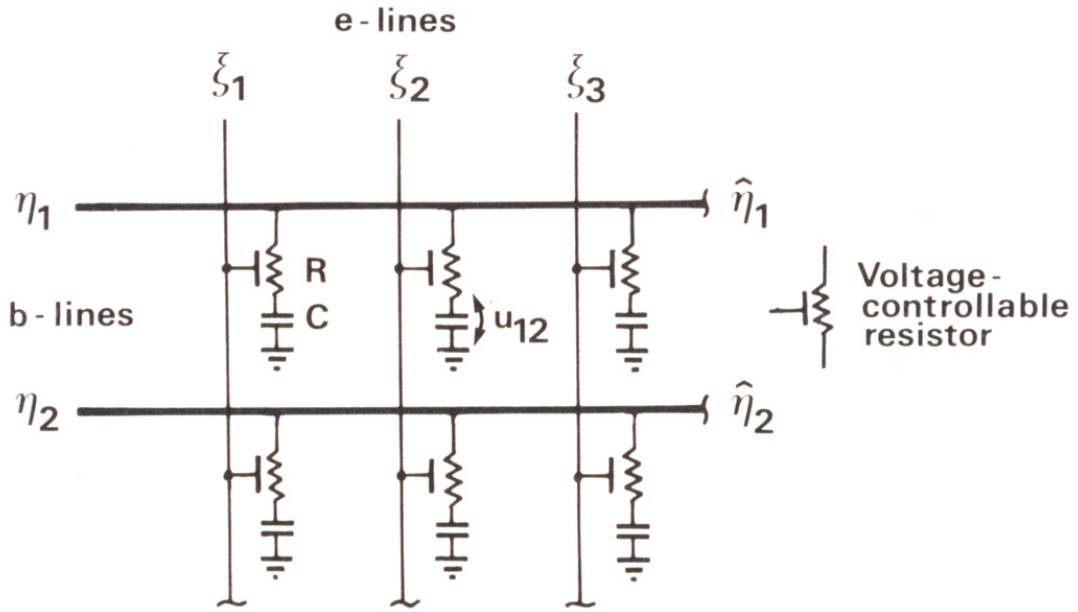


Abb. 11: Lernmatrix nach STEINBUCH

Für die aus den 1960er-Jahren stammende Lernmatrix nach STEINBUCH[32] wird bei KOHONEN eine Hardwarerealisierung angegeben, die als Vorbild für ande-

[32] vgl. Karl Steinbuch: „Automat und Mensch“, Springer 1963, oder [Hecht-Nielsen 91, Kap. 4.3]

re Realisierungen matrixförmig aufzubauender neuronaler Netze dienen kann.[33] STEINBUCH entwickelte diese Lernmatrix zur Simulation der Klassischen Konditionierung (s. Pawlowscher Hund). An den „e-lines“[34] wird das Fragetupel angelegt, an den „b-lines“[35] das Antworttupel. Die Leitungen nehmen nur die beiden Zustände „0“ oder „1“ ein, zwei Zustände, denen jeweils ein elektrisches Potential zugeordnet ist, der „1“ das höhere.

Transistoren (z.B. ein im Sperrzustand hochohmiger FET) werden als „voltage-controllable resistor“ eingesetzt. Sie schalten bei „1“ auf der zugehörigen e-line durch und es fließt in Abhängigkeit von R (typischerweise etwa 1 kΩ) und dem Zustand der b-line aus dem Kondensator Ladung ab (s. Entladekurve eines RC-Glieds), sonst bleibt die Ladung erhalten (bis auf die üblichen Leckagen). Je öfter die jeweilige e-line beim Lernen eine „1“ liefert, desto öfter kann die Entladung der Kondensatoren geschehen.

Das Abfragen der Lernmatrix geschieht über die e-lines, auf den b-lines stellen sich die aus den verbliebenen Kondensatorladungen ergebenden Werte ein.

Der Vorteil dieser Hardwarelösung liegt darin, dass mit einfachen Mitteln synaptische Verbindungen hergestellt werden, deren Verbindungsstärke sich in Vielfachen der Elementarladung ausdrückt, also viele verschiedene Werte annehmen kann.

In Kapitel 4.1 treten zum Aufbau der VIDAS -Hardware an die Stellen der RC-Glieder bistabile Kippstufen.

3.2 PAN-Systeme I bis IV

PALM begründet in „Parallel Processing for Associative and Neuronal Networks“[36] den Bedarf für eine Hardwarelösung von Assoziativspeichern damit, dass die Rechenzeit auf gewöhnlichen Rechnern quadratisch mit der Größe der Matrix zunimmt. Eine Abfrage eines 100-Neuronen-Netzwerks benötigte seinerzeit 2 s. Daher schlägt er eine parallele Abarbeitung von Matrixspaltenbündeln vor, die jeweils von einem Zentralprozessor geleistet wird (PAN-System).

Das erste solche PAN-System[37] ließ acht Z80-Prozessoren mit einem 32 KB-RAM-Speicher arbeiten, in welchem außer der Matrix auch Arbeitsspeicher für die acht Z80 untergebracht wurde, so dass die Assoziativmatrix eine Größe von 448 x 448 Synapsen bekam. Ein zusätzlicher Z80 in einem modifizierten Sinclair ZX 81

[33] s. [Kohonen 88, S. 74-77, 79]
[34] e — Eigenschaften
[35] b — Bedeutungen
[36] vgl. [Palm84, S. 201]
[37] PAN — Parallel Associative Network

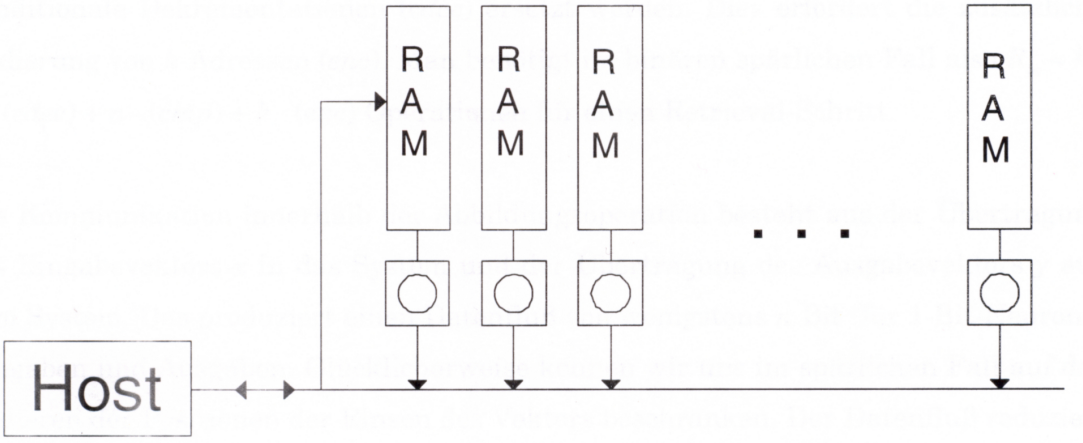


Abb. 12: Aufbau eines PAN-Systems

regelte den asynchronen Betrieb der acht Z80. In Abb. 12 sind die Prozessoren unterhalb der RAM-Spalten als Quadrate mit einer Kreislinie darin dargestellt.[38] Im System PAN I war jeder Z80 für 448 / 8 = 56 Matrixspalten zuständig. Die Z80 arbeiteten mit einem Systemtakt von 4 MHz. Als Host fungierte eine DEC PDP 11. Demgegenüber erhielt PAN II 22 Prozessoren für eine Assoziativmatrix bis zur Größe von 3168 x 3168 Synapsen. Die Assoziativmatrizen brachte man danach in VLSI-Technik auf einem Chip unter, der BACCHUS getauft wurde. Diese Bacchus-Chips enthalten 32 8-Bit-Zähler, durch die die Summen in den Dendriten gebildet werden. Der Einsatz von Z80-CPUs entfällt dadurch. Für den Aufbau von PAN III wurden 16 Bacchus-Chips eingesetzt, wodurch bis zu 512 Nervenzellen mit je 2 K Synapsen von einem PC-AT aus genutzt werden konnten.

PALM berichtet in „The PAN System and the WINA Project“[39] vom System PAN IV. In diesem System arbeiten 4 K Prozessoren für eine Matrix mit 4096 Nervenzellen und je 256 K Synapsen. PAN IV zielt auf das Anreihen von noch größeren Anzahlen an weiterentwickelten BACCHUS-Chips.[40] Eine PAN IV-Karte ist mit 8 Bacchus-Chips bestückt, wodurch 256 Nervenzellen mit je 256 K Synapsen entstehen. Die Breite des PAN-Busses, an dem die PAN IV-Karten angeschlossen werden, erlaubt bei 32 Bit den Umgang mit bis zu 4 G Nervenzellen. Wenn in Abb. 13 ein Maximum von 19 Karten angegeben wird (das wären ja „lediglich“ 19 x 8 x 256 = 38.912 Neuronen), dann bezieht sich das auf die technischen Mög-

[38] aus [Holthausen 94, S. 16]

[39] s. [Palm 93]

[40] HOLTHAUSEN erwähnt in [Holthausen 94, S. 19] die Variante BACCHUS III, an die nahezu beliebige DRAM-Bausteine anschließbar seien.

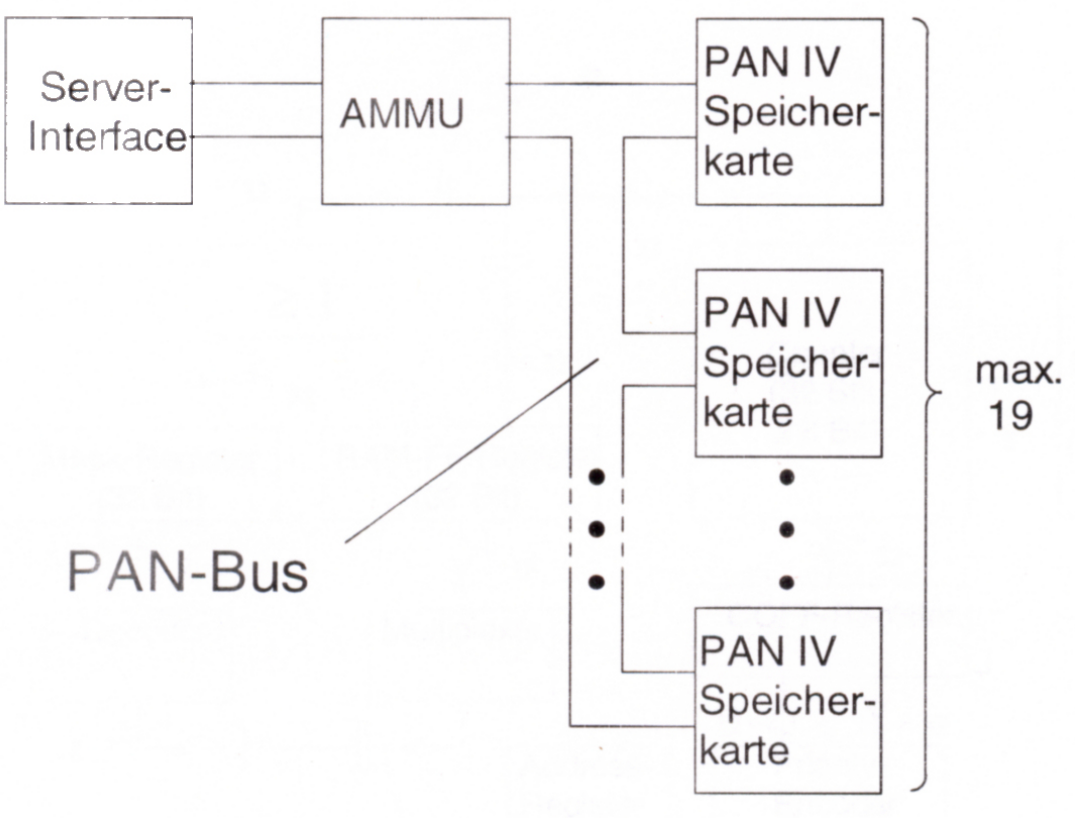


Abb. 13: Das PAN IV-System

lichkeiten eines einzelnen Bussystems,[41] welches die PAN IV-Karten aufnimmt. Doch es ist möglich, das Bussystem zu verlängern, so dass man zu der von PALM angegebenen maximalen Größenordnung gelangt.

Die „Associative Memory Management Unit (AMMU)“ stellt das Bindeglied zwischen dem PAN IV-System und dem anwendenden Computersystem dar. In ihm arbeitet ein Motorola 68030-Prozessor. Ein PAN-Server regelt den Zugang zum System und stellt eine TCP/IP-Anbindung her, um den Netzwerkzugriff auf das PAN-System zu ermöglichen.

Diese in Einzelheiten wiedergegebenen Anstrengungen für eine Hardwarelösung zu einer Assoziativmatrix mögen aufzeigen, auf welchen Wegen man letztlich zu einem komplexen System gelangt, dessen Kosten und dessen aufwändiger werdende Zugriffsverwaltung zu rechtfertigen sind. HOLTHAUSEN stellt sich 1994 in „Ein Vergleich verschiedener Implementationen binärer neuronaler Assoziativspeicher“ die Frage, ob die Geschwindigkeit des PAN IV-Systems den Vergleich

[41] VME-Bus-Backplane, s. [Holthausen 94, S. 19]

mit Softwaresimulationen auf Universalrechnern mit von Neumann-Architektur standhält. Er kommt zu dem Ergebnis, dass PAN IV den damaligen schnellen Rechnern (SPARCstation 10/40) ab einer gewissen Matrixgröße zeitlich überlegen war.

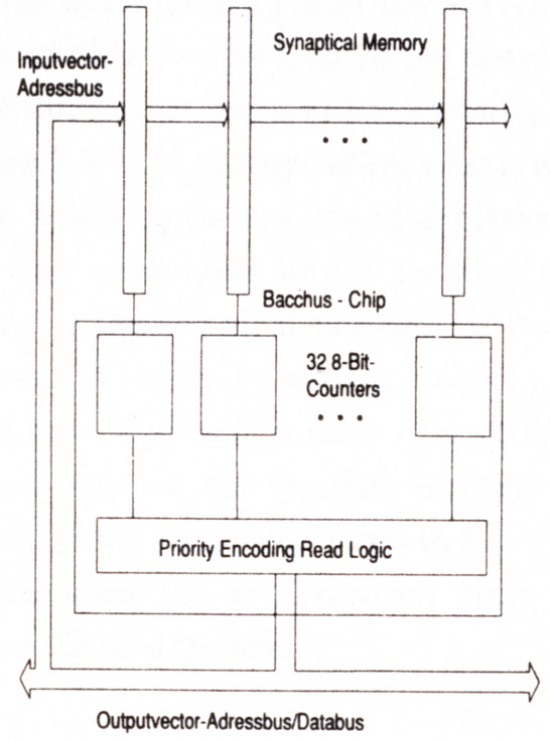


Abb. 14: BACCHUS-Chip

Der Bacchus-Chip (s. Abb. 14) wird bei Siemens in einem 1,5 μm-CMOS-Prozess gefertigt, hat an seinem Gehäuse 68 Anschlüsse, enthält 26.159 Transistoren und wird mit einer Taktfrequenz von 10 MHz betrieben.[42] Beim Lernen werden die Einträge eines Antworttupels Byte für Byte auf den 8-Bit-Datenbus des Bacchus-Chips übertragen und dieser leistet dann gegebenenfalls das Setzen von Einsen in der Matrixzeile, die durch das Fragetupel im zugeordneten RAM adressiert ist. Beim Abfragen werden die Zähler in den Bacchus-Chips auf den gewünschten Schwellenwert θ gesetzt und anschließend beim Auftreten einer „1“ in den durch das Fragetupel adressierten Matrixzeilen der zur betreffenden Matrixspalte gehörende Zähler dekrementiert. Diejenigen Zähler, die während der Abfrage auf Null springen, stellen durch ihre Position die Einsen des Antworttupels dar. PALM erwähnt in [Palm 93, Kap. 1] „In our first implementations of the PAN system

[42] s. [Holthausen 94, S. 26]

we have restricted ourselves to the case of binary values, i.e. we assume that the components X_j^i and Y_j^i of input and output vectors are in $\{0, 1\}$." Diese Beschränkung erlaubt jedoch den eben beschriebenen Einsatz einfacher Digital-Zähler in den Bacchus-Chips statt aufwändigerer Addierwerke. Die VIDAS -Maschine setzt solche Digital-Zähler ebenfalls ein,[43] allerdings in inkrementierender Weise, wie das Kapitel 4.4 zur Schwellwertlogik zeigt.

Im wesentlichen Unterschied zum PAN-System arbeitet die VIDAS -Maschine auch die anwendenden Programme im Assoziativspeicher ab. Während beim PAN-System das die Assoziativmatrix nutzende Programm auf einem Rechner außerhalb des Systems abläuft und die Frage- und Antworttupel über die AMMU in die Matrix hinein- und herausgeschafft werden müssen, bleiben die Daten bei der VIDAS -Maschine innerhalb des Systems und werden dort verarbeitet. Damit ist ein Schritt getan, den 1945 schon John VON NEUMANN machte, als er Programme zusammen mit ihren Daten in den gleichen Speicher legte.[44]

3.3 Hardware für andere Neuronale Netze

RIEMSCHNEIDER gibt in „Parallele Hardware für Backpropagation-Netze auf der Basis stochastischer Rechenwerke"[45] eine Übersicht über die Bemühungen, für in Schichten (layer) organisierte Neuronale Netze geeignete Hardware zu entwickeln.

Aus der Tabelle in Abb. 15 wird deutlich, dass es für diese Art neuronaler Netze auf Festkomma-Rechenwerke zur schnellen Berechnung der Gewichte und auf die flexible Verschaltbarkeit der Neuronen ankommt. In CPS (connections per second) wird die Geschwindigkeit in der Abfragephase des Netzes angegeben (Verbindungsberechnungen pro Sekunde), in CUPS (connection updates per second) die Geschwindigkeit in der Lernphase des Netzes (Anpassungsberechnungen für die Gewichte gemäß Lernregel pro Sekunde).

Um einen Eindruck von der Funktionsweise und den Geschwindigkeiten eines Backpropagation-Netzwerks zu erhalten, wurde im Rahmen des VIDAS -Projekts ein solches auf verschiedenen Rechnern implementiert, zuletzt 2004 auf einem Pentium IV-System mit einer Taktfrequenz von 3,2 GHz. Die Anzahl der Schichten und die Anzahl der Nervenzellen pro Schicht sind im Prinzip frei wählbar. Unten rechts erkennt man in Abb. 16 das Lernmuster (Trainingsmuster), links davon die Darstellung der Ausgabeschicht des Netzwerks, darüber die Darstellung

[43] vgl. Kap. 4.2

[44] John von Neumann: „First Draft of a Report on the EDVAC" (1945): Es ging ihm darum, zur Laufzeit des Programms bedingte Sprünge ausführen zu können.

[45] s. [Riemschneider 96, S. 11]

Produkt/Projekt	Leistung/Netzmodelle	Technologie	Einordnung/Bem.
Hardwareunterstützung des Trainingsvorgangs			
Siemens 1994 **Synapse-1** < 64M Synapsen 16 bit Festkomma keine feste Neuronen/-Synapsenanzahl	5.1 GCPS ca. 1 GCUPS div. Netzmodelle versch. Lernverfahren	insges. ca. 2200 Chips davon: 8×MA16 je 187mm^2, 1.0μm 25...40 MHz	Gruppe 2 [154, 168] **parallele Vektor-Matrixop.** Firmware, Netzstruktur in Software
Hitachi 1993 **WSI-Neurocomputer** 72K Synapsen (16 bit) 1152 Neuronen (9 bit) Festkomma	4 (?) GCPS ca. 2 GCUPS Verdoppeln der Neuronenzahl im Feed-forward-Modus	8 × 5 Zoll-Wafer 0.8μm 144 Neuronen/Wafer 10 MHz	Gruppe 3b [200] **Neuronen-parallel** Hardware-Backpropagation
kein Trainingsvorgang in Hardware			
Neural Semiconductors 1990 **NU32-SU3232** 1K Synapsen (8 bit) 32 Neuronen, unipolare zufällige Bitströme	1.2 GCPS (60 GCPS on-board) -- CUPS	2 Chiptypen bis zu 50 Chips skalierbar 25 MHz	Gruppe 4a [191, 192] **Synapsen-parallel** Training in Software
Intel Corp. 1992 **ETANN 80170NX** 10K Synapsen ≈7 bit analog 64 Neuronen	2 GCPS — CUPS Feed-forward-Netze	1 Chip (2.0μm) Floating-Gate-Speicher analog ext. Signale 300K Pattern·s^{-1}	Gruppe 5a [135, 14] **Synapsen-parallel** Trainingskit Software

Abb. 15: Überblick über Neurohardware 1996

der Klassifizierungsleistung der Nervenzellen der hidden layer (hier drei Schichten à fünf Nervenzellen). Nach dem Lernen wurden alle Schichten mit allen Pixeln der Eingabeschicht abgefragt. Die Ergebnisse sind in Abb. 16 für alle beteiligten Nervenzellen dargestellt. Die Ergebnisse weichen trotz gleicher Trainingsdaten immer wieder voneinander ab, je nachdem, welches lokale Optimum beim Lernen gefunden wurde.

Das Reizvolle an Neuronalen Netzen (seien es als Matrizen oder in mehreren Schichten organisierte) findet sich in Abb. 16 deutlich: auch auf Fragen, die zu beantworten nicht gelernt wurde, geben die Netze dennoch meist eine akzeptable Antwort.

Als Nachteil des Backpropagation-Netzwerks erfährt man die relativ langen Lernzeiten, die vergehen, bis sich die Gewichte dank der Lernregel so eingestellt haben, dass das Netz bei Abfragen die gewünschten Ergebnisse liefert.

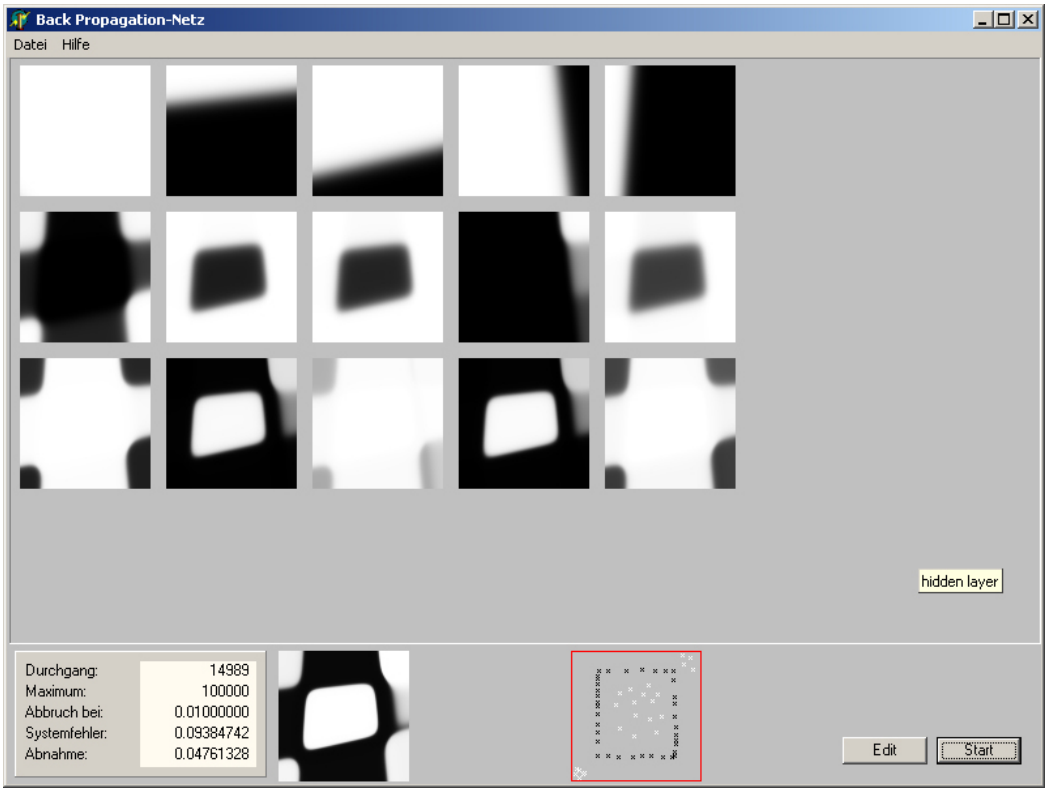


Abb. 16: Die klassifizierende Leistung der hidden layer eines Backpropagation-Netzwerks

Ob durch einzelne in Schichten (layer) angeordnete Nervenzellen oder ob durch Zellverbände in Matrixstruktur die Nachbildung von assoziativer Gehirnleistung besser gelingt, beantwortet der Artikel „Cell Assemblies as a Guideline for Brain Research“ von PALM.[46] Er favorisiert den Zellverband und misst damit einer einzelnen Nervenzelle eine geringe Bedeutung bei. Das passt zur Beobachtung, dass im Gehirn täglich Tausende Nervenzellen absterben.[47] Wenn man also der einzelnen Nervenzelle und ihren synaptischen Verbindungen in einer Simulation weniger Gewicht geben müsste, wird man sich fragen, ob der Nachbildung einer einzelnen Nervenzelle eines Schichtennetzes mit ihren durch sechzehnstellige Festkommazahlen dargestellten Verbindungsgewichten[48] nicht „zu viel“ an Leistung zugesprochen wird, so dass ihr möglicher Ausfall eine zu große Wirkung nach sich zöge. Wegen dieser prinzipiellen Bedenken und wegen des erheblichen Zeitaufwands beim Lernen und bei der Auswahl der Trainingsdaten kam diese Art neuronaler Netze für den Aufbau einer VIDAS -Maschine nicht in Betracht.

[46] vgl. [Palm 90]
[47] s. http://www.oncosite.de/51.0.html
[48] s. „Synapse-1“ in Abb. 15

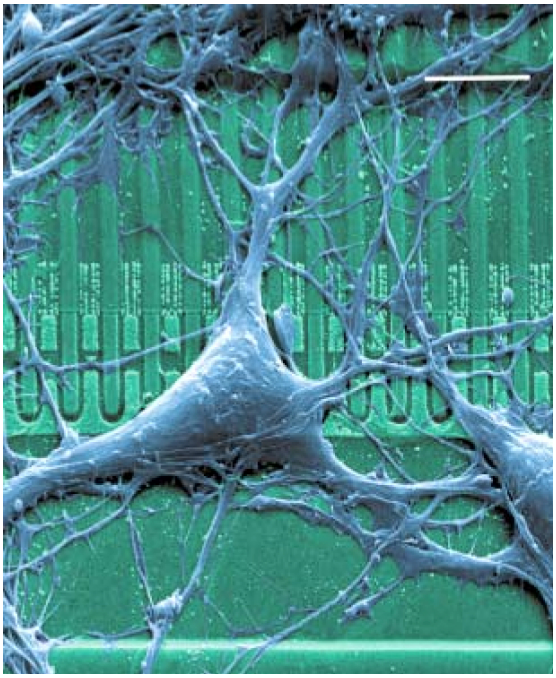


Abb. 17: Rattennervenzelle auf Siliziumchip

Peter FROMHERZ vom Max-Planck-Institut für Biochemie in Martinsried entwickelte einen „Neuro-Chip“,[49] indem er eine Kopplung zwischen Nervenzellen und Halbleitern herstellte (s. Abb. 17). Der Skalierungsbalken oben rechts im Bild hat eine Länge von 10 μm. Die Kopplung erfolgt über die elektrischen Felder der Kopplungspartner in einer Weise, dass möglichst weder der Halbleiter korrodiert noch die Zelle durch elektrochemische Prozesse geschädigt wird. Halbleiter und Nervenzelle werden dazu auf einen Abstand von 8 μm gebracht. FROMHERZ bezeichnet sein Vorgehen als Grundlagenforschung und nennt die Absicht, ein „einfaches, chipkontrolliertes neuronales Netz aufzubauen“. Mit 128 x 128 Kontaktsensoren auf einer Fläche von etwa einem Quadratmillimeter können 32 Millionen Zellsignale pro Sekunde aufgezeichnet werden. Eines dieser Zellsignale äußert sich dabei in einer zu messenden Spannung im Bereich von 10^{-6} bis $5 \cdot 10^{-3}$ Volt. Diese Angaben mögen verdeutlichen, welcher Aufwand zum Aufbau neuronaler Netze aus diesen Neuro-Chips gehört. Dennoch gelang Bernhard WOLF die Kopplung von Riechzellen einer Zecke mit elektronischen Schaltkreisen, um einen Ortungssensor für Menschen zu erhalten. Er nennt als Lebensdauer für diese Biosensoren einige Tage, da auch die Stoffwechselprodukte der Zellen die Kontakte

[49] vgl. Peter Fromherz: „Interfacing von Nervenzellen und Halbleiterchips — Auf dem Weg zu Hirnchips und Neurocomputern?“, in: „Physikalische Blätter“ 57 (2001) Nr. 2

zur Mikroelektronik verschlechterten.

Den Ansatz, statt einzelner Nervenzellen gleich einen ganzen Zellverband auf ein Sensorfeld zu verbringen, ging in Folge nicht nur FROMHERZ,[50] sondern auch Thomas DEMARSE, der einer Ratte 25.000 lebende Nervenzellen entnahm und mit Hilfe von 60 Elektroden eine Verbindung mit einem Flugsimulator herstellte.[51] Das Flugzeug sei von dem Nervennetz mit fortschreitender Versuchsdauer immer besser kontrolliert worden.

Damit kommt man dem Anliegen von PALM entgegen, das Studium von Zellverbänden voranzutreiben. Der Aufbau einer fehlertoleranten Maschine aus solchen Neuro-Chips ist denkmöglich, wenngleich die Voraussetzungen in der Praxis dafür noch nicht geschaffen sind.

[50] vgl. heise online in „ISSCC: Chip mit Hirn“ vom 8.2.05: „Auf einer Matrix von 128x128 Sensor-Transistoren wurde eine Schicht von Rattenneuronen platziert.“

[51] vgl. heise online in „Petrischale als Pilotensitz“ vom 24.10.04

4 VIDAS -Hardwarelösung

Zur Entwickung einer Hardwarelösung für eine robuste, frei programmierbare Maschine aus Assoziativmatrizen werden die Matrizen mit Hilfe von Flip-Flops aufgebaut (Kap. 4.1) und zur gewünschten Größe zusammengesetzt (Kap. 4.2). Während das Lernen von Mustern in einem einzigen Taktschritt erfolgen kann, wird in der auf Digitaltechnik beruhenden Lösung das Abfragen der Matrix über ein Schieberegister vorgenommen, was die Abfragezeit linear von der Anzahl der Matrixzeilen abhängig macht (Kap. 4.3). Die Auswertung der bei den Matrixabfragen entstehenden Schwellwerttupel erledigt die Schwellwertlogik automatisch und liefert dabei auch die Schwellwertgüte (Kap. 4.4). Bevor die einzelnen Teile zur VIDAS -Maschine zusammengesetzt werden (Kap. 4.9), erfolgen umfangreiche Software-Erprobungen durch die Projekte VIDAS 1, 2, 3 und 4 (Kap. 4.5 bis 4.7). Die Ergebnisse der Tests finden sich beim Zusammensetzen von Matrix mit Schwellwertlogik (Kap. 4.8) und beim Aufbau der Maschinen VIDAS 7 und VIDAS 9 wieder (Kap. 4.9). Programme für die Maschinen werden durch Programmeditoren erfasst und aufbereitet, so dass den Maschinen schließlich die Belegungen für die Matrizen zur Verfügung stehen (Kap. 4.11). Zur Abarbeitung der Programme stehen das Assoziieren einer Programmzeile mit ihrer Nachfolgerin (Kap. 4.10.2) und die Ausführung der damit verbundenen Befehle (Kap. 4.10.1) im Mittelpunkt der VIDAS -Maschine.

4.1 Nachbildung synaptischer Verbindungen

Stellt man sich an den Stellen, an denen in der von KOHONEN vorgestellten Hardwarelösung für die STEINBUCHsche Lernmatrix[52] das RC-Glied sitzt, eine bistabile Kippstufe vor, so erhält man eine ungleich einfachere Lösung für ein neuronales Netz in Matrixstruktur, allerdings ohne Option auf Verbindungsgewichte außerhalb von $\{0, 1\}$. Aus Abb. 18 wird ersichtlich, wie dafür bei der VIDAS -Maschine RS-Flip-Flops als bistabile Kippstufen eingesetzt werden.

Die horizontalen Axone sind hier ihrer Funktion gemäß mit „Lies (RD)“ bezeichnet, die vertikalen mit „Schreibe (WR)“. Liegt auf beiden Leitungen eine „1“ wird das RS-Flip-Flop gesetzt (der Ausgang „Q“ geht auf „1“-Potential), liegt (in der Abfragephase) nur am horizontalen Axon eine „1“ an, wird auf das Ausgangs-Axon „Ergebnis (OUT)“ der Zustand von „Q“ gelegt. Mit einer „1“ auf den „Lösche (CLR)“-Leitungen wird das RS-Flip-Flop zurückgesetzt.

Ein RS-Flip-Flop lässt sich aus zwei Transistoren T_1 und T_2 aufbauen, wie Abb. 19 zeigt. Geht der Eingang S auf „1“-Potential, schaltet T_1 durch und $\overline{Q}$ gelangt

[52] s. Kap. 3.1

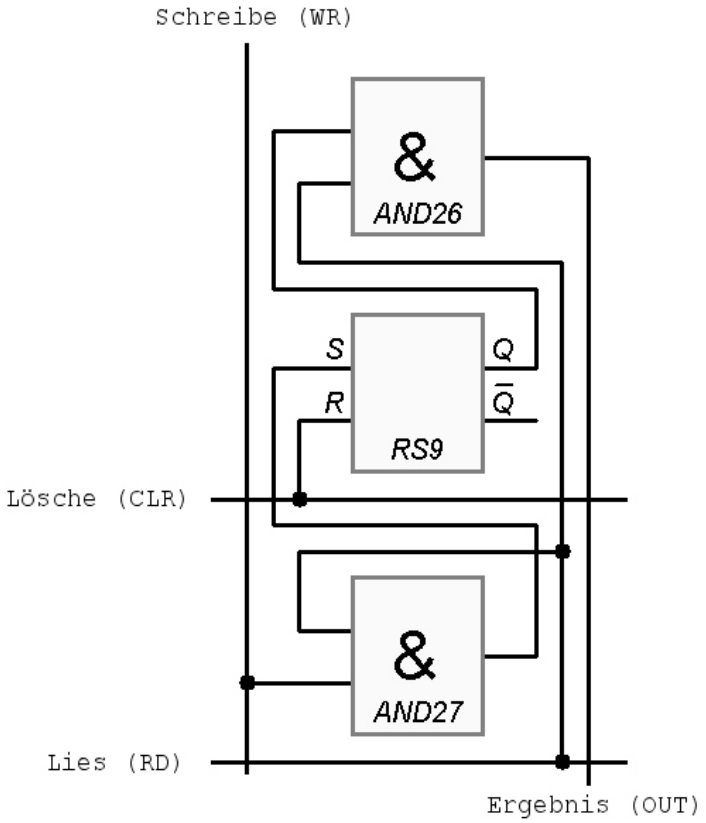


Abb. 18: Bistabile Kippstufe als synaptische Verbindung

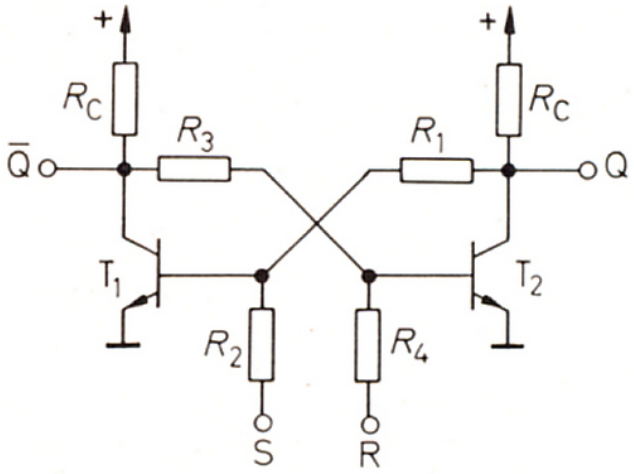


Abb. 19: Bistabile Kippstufe durch Transistoren

auf niedriges Potential, wodurch wiederum T_2 sperrt und Q auf „1“-Potential hebt.[53] Die eingezeichneten Widerstände R_i dienen dazu, die fließenden Ströme zu begrenzen, um die Transistoren nicht zu zerstören. Eine „1“ an Eingang R dient zum Zurücksetzen (Reset) des RS-Flip-Flops, was bedeutet, dass spiegelbildlich zum eben beschriebenen Vorgang des Setzens des S-Eingangs (Set) die bistabile Kippstufe einen Zustand einnimmt, in welchem Q auf „0“-Potential und $\overline{Q}$ auf „1“-Potential liegt.

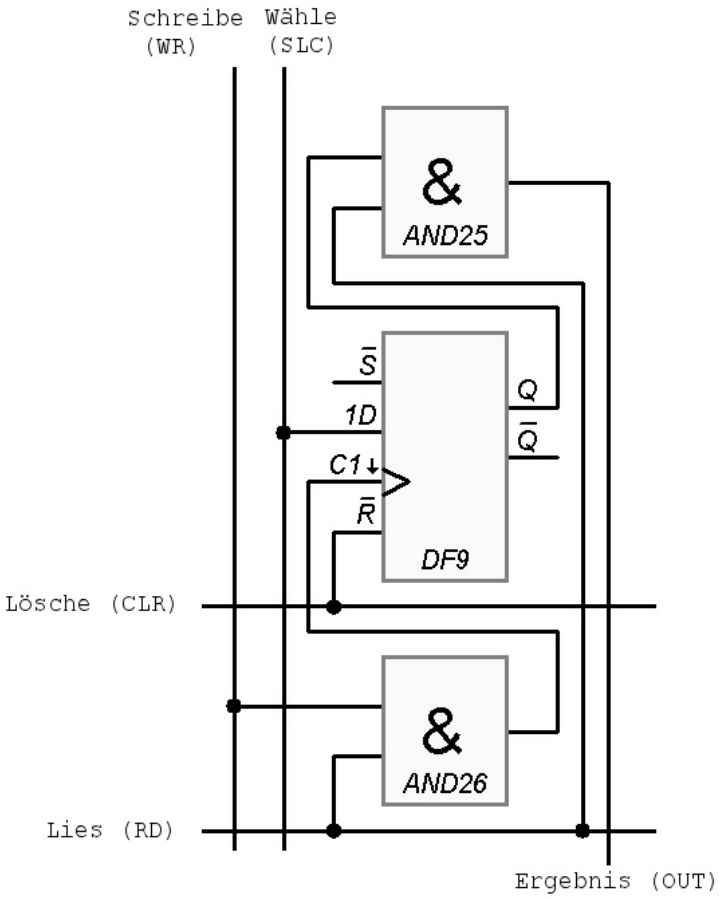


Abb. 20: JK-Flip-Flop als synaptische Verbindung

Interessant sind die Auswirkungen des Einsatzes anderer Flip-Flops an der Stelle, an der in der STEINBUCHschen Lernmatrix bei KOHONEN das RC-Glied angegeben wird. In Abb. 20 bekommt ein JK-Flip-Flop diesen Platz.

Liegt an den Axonen „Schreibe (WR)“ und „Lies (RD)“ das „1“-Potential, dann wird dem JK-Flip-Flop ein Signal an $C1$ geliefert, auf dessen negative Flanke hin

[53] vgl. [Tietze/Schenk 02, S. 601]

der Zustand von „Wähle (SLC)“ an $1D$ auf den Ausgang Q kopiert wird. Also kann auf diesem Wege auch eine „0“ in die Synapse eingetragen werden, was beim oben geschilderten Einsatz von RS-Flip-Flops nicht möglich ist. Das Abfragen und Löschen solcher Assoziativmatrix geschieht hingegen wie oben.

Assoziativmatrizen mit solchen JK-Flip-Flops dienen dazu, in VIDAS mit Variablen umzugehen, die bei einer Wertzuweisung ihren alten Wert „vergessen“ können müssen. PALM beschreibt in [Palm 90, S. 140] „schnelle Hebb-Synapsen“, wie sie VON DER MALSBURG vorschlägt. PALM nutzt diesen Synapsentyp, um einen natürlichen Mechanismus anzugeben, durch den sich die Nervenzellen zu einem Zellverband vereinen.[54] Diese Idee einer schnellen Hebb-Synapse wird durch den Einsatz der JK-Flip-Flops aufgegriffen, indem dadurch ein Nervenzellenbündel, welches gerade noch für eine Variable einen Wert lieferte, „aufgelöst“ werden kann, um einem Nervenzellenbündel den Vortritt zu lassen, welches der Variablen einen neuen Wert zuordnet.

4.2 Matrixaufbau

1989 wurde die Hardwarelösung zur Assoziativmatrix mit RS-Flip-Flops aus Kap. 4.1 mit Hilfe eines Digitalsimulators erstmals vorgestellt (s. Abb. 21).[55] [56] Doch erlaubte die Leistungsfähigkeit des Simulations-Rechners damals keinen komplexen Ausbau und fruchtbaren Test des Systems und es blieb bei der Simulation der Vorgänge in einer einzelnen 8 x 8-Matrix.[57]

In Abb. 21 wird ein Dendrit (eine Spalte einer Assoziativmatrix) in der durch Abb. 18 gegebenen Technik angezeigt. Durch ein ODER-Gatter wird das Axon Q (oben) bei der Abfrage auf „1“-Potential gelegt, falls das über die Anschlüsse A_i adressierte Flip-Flop gesetzt ist. Beim Lernen wird die Matrixspalte durch eine Eins am „Chip Select (CS)“-Eingang ausgewählt und die Flip-Flops erhalten durch einen Impuls über den „Write (WR)“-Eingang diejenigen Einsen, die an den Anschlüssen A_i liegen. Das Zurücksetzen der Flip-Flops der Matrixspalte erfolgt über den „Clear (CLR)“-Eingang.

Dank schnellerer Rechner und leistungsfähigerer Digitalsimulatoren sind jetzt

[54] Er schreibt a.a.O: „A natural mechanism for this would be fast Hebb synapses which are strengthened very quickly between co-active neurons and which also decay very quickly when the neurons are no longer active.“

[55] Die erste Version vom 9.6.88 entstand mit 4 x 4 Synapsen nur auf dem Papier.

[56] s. [Dierks 89a]

[57] Als Digitalsimulator wurde 1989/90 „ST-Digital“, Version 2.0, des Heim Verlags, Darmstadt, auf einem Atari ST 1040 eingesetzt.

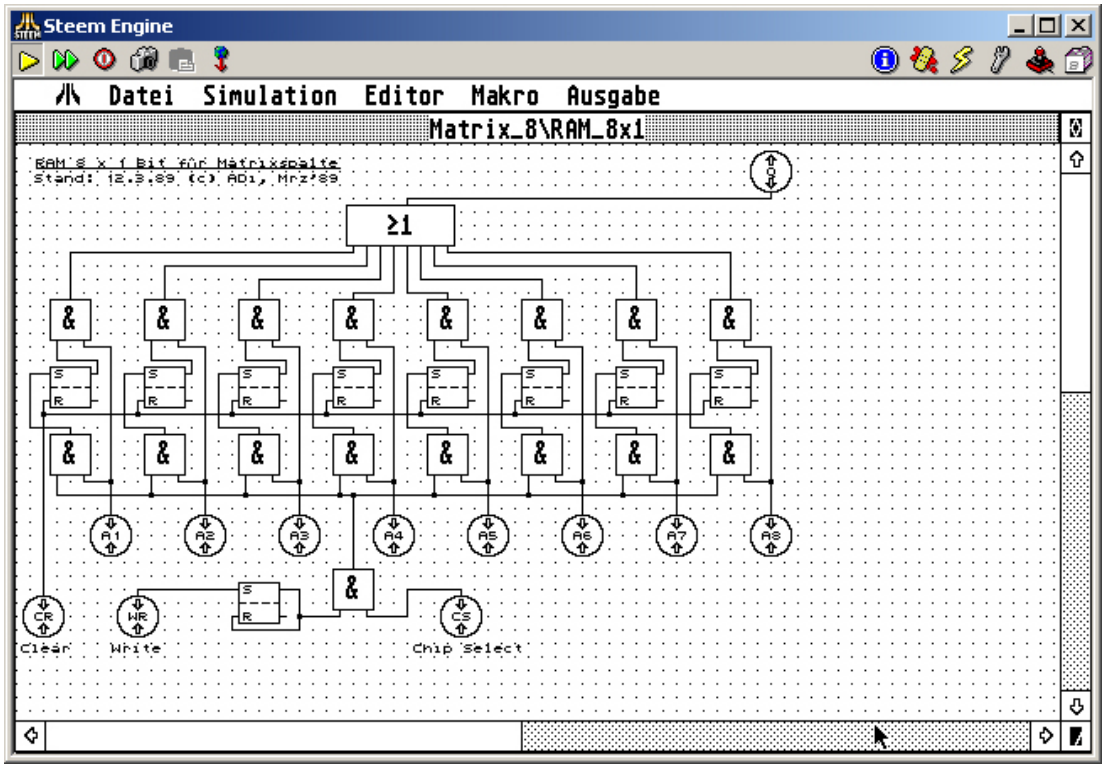


Abb. 21: Hardware-Spalte einer Assoziativmatrix mit ST-Digital

deutlich aufwändigere Simulationen möglich.[58] [59] Der Digitalsimulator ProfiLab erlaubt dem Entwickler, der auch auf Anschauung und Abstraktion angewiesen ist, die Definition von Makros, die Anzeige des Zustands der Leitungen, einen Frontplattenentwurf und eine Bedienung der Schaltung über diese Frontplatte, den Einsatz von Logikanalysatoren, den Zugriff auf Schnittstellen des Rechners, um gegebenenfalls externe Hardware anzusteuern.

Im Jahr 2004 wurde durch ProfiLab VIDAS 7 fertiggestellt (aus fünf 8 x 8-Assoziativmatrizen), 2005 dann VIDAS 9 (aus sechs 16 x 16-Assoziativmatrizen). Für VIDAS 9 muss ProfiLab das Verhalten von gut 1.500 Flip-Flops simulieren. Abbildung 22 zeigt daraus einen Ausschnitt.

Der Übergang von 8 x 8-Matrizen auf 16 x 16-Matrizen oder auf noch größere Matrizen gerät einfach, da man größere aus kleineren Matrizen beliebig in Breite und Höhe zusammenstecken kann (s. Abb. 23).[60] So ist es dann wiederum auch

[58] Auch wenn der Start der VIDAS 9-Simulation zurzeit noch immer etwa 15 Minuten dauert.

[59] Als Digitalsimulator wurde ab 2004 „ProfiLab“, Version 3.0, des Ingenieurbüros Abacom in Ganderkesee auf einem Pentium 4-System mit gut 3 GHz Systemtaktfrequenz genutzt.

[60] Eine größere Abbildung befindet sich im Anhang in Abb. 88

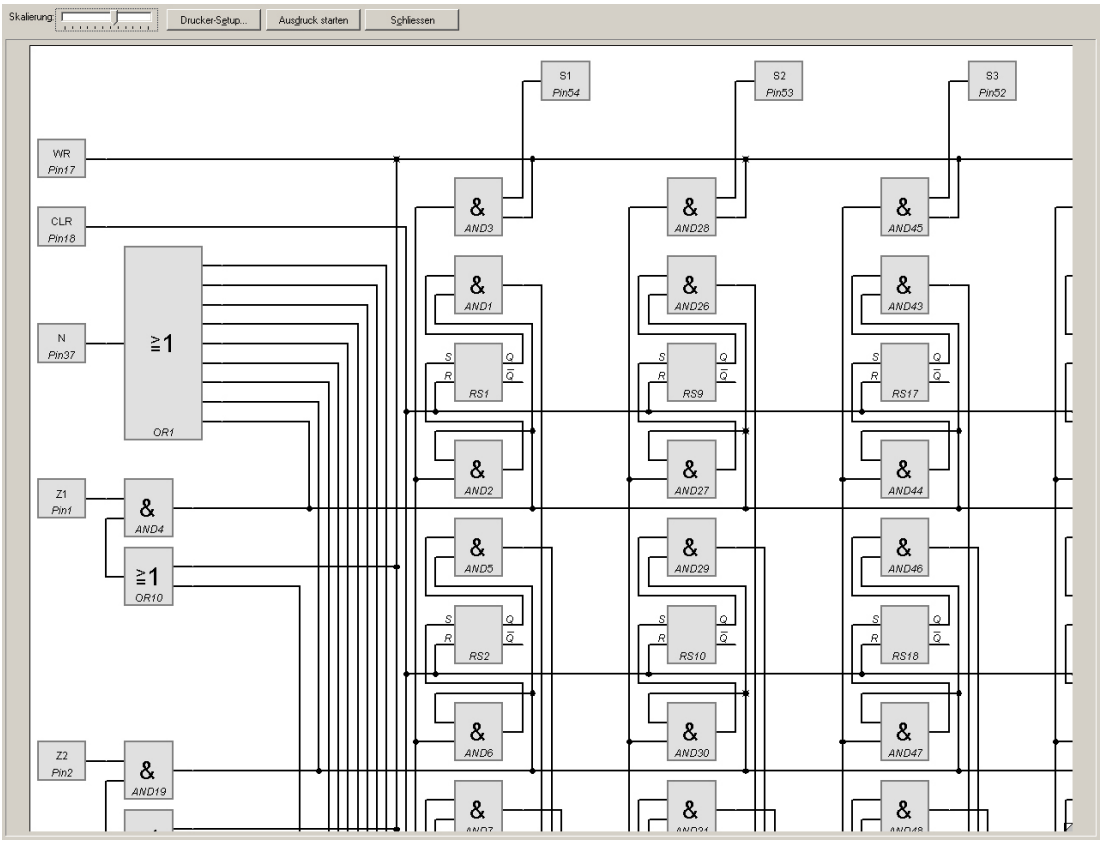


Abb. 22: Mehrere Spalten einer Assoziativmatrix mit ProfiLab

möglich, mehrere Matrizen der VIDAS -Maschine zusammenzusetzen, denen bei der Speicherung und Abarbeitung von Programmen verschiedene Aufgaben zukommen.[61]

Für den Aufbau von Matrizen aus den mit JK-Flip-Flops dargestellten synaptischen Verbindungen (s. Kap. 4.1) gilt das vorstehend Beschriebene ebenfalls.

Für die Matrixspalten könnte man wie im PAN-System[62] auch marktgängige Speicherbausteine einsetzen. Diese sind dann üblicherweise byteweise adressierbar, so dass man vor der Wahl steht, entweder von jedem Byte nur ein Bit zu nutzen oder wie im BACCHUS-Chip[63] eine zusätzliche Adressierlogik einzusetzen, die den bitweisen Zugriff auf den marktgängigen Speicherbaustein ermöglicht. Im ersten Falle verliert man Platz, im zweiten Falle Zeit. Eine weitere Möglichkeit wäre der Einsatz bitweise adressierbarer Bausteine der ursprünglich aus der

[61] vgl. Kap. 4.9, Matrizen A, B, P1, P2
[62] vgl. Kap. 3.2
[63] vgl. Abb. 14

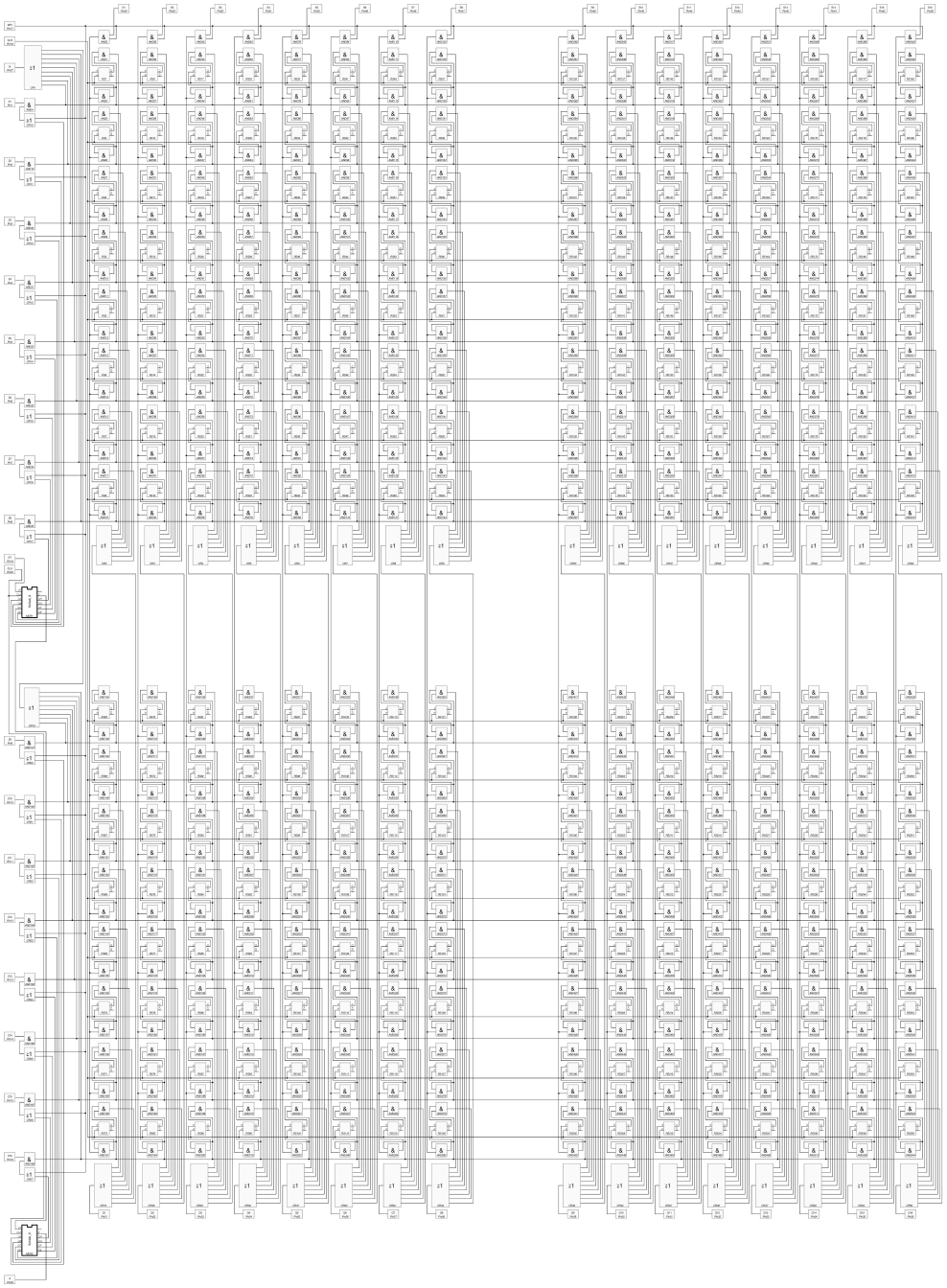


Abb. 23: Eine 16 x 16-Matrix wird aus vier 8 x 8-Matrizen zusammengesteckt.

TTL-Technik bekannten Art 7481.[64]

[64] 16-Bit-RAM (16 x 1), vgl. „TTL-Taschenbuch“, IWT-Verlag, Vaterstetten

4.3 Eintragen in und Abfragen der Matrix

Das Eintragen eines Musters in eine wie in Kap. 4.2 verwirklichte Assoziativmatrix kann in nur einem Systemtakt erfolgen.[65] Hier zeigt sich ein besonderer zeitlicher Vorteil gegenüber einer Softwarelösung, die je nach Implementationsform der Matrix einen Zeitaufwand in der Größenordnung von mindestens O(n) mit sich bringt, mit n als Anzahl der Matrixzeilen.

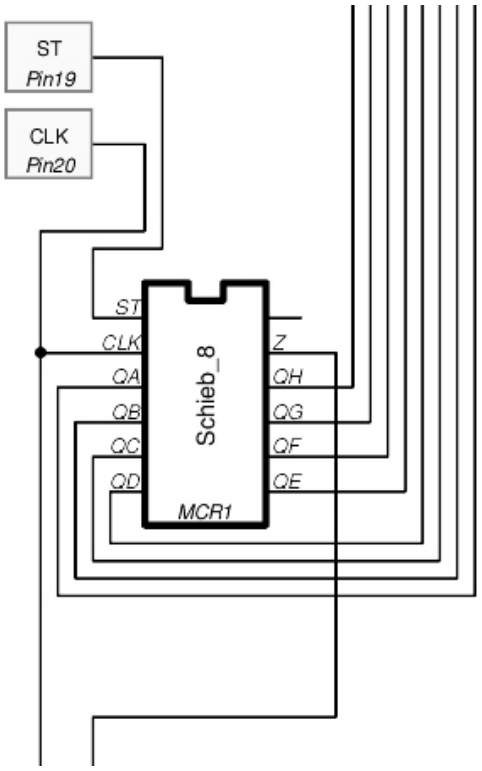


Abb. 24: Schieberegister zur Matrixabfrage

Innerhalb dieses einen Systemtaktes sind Frage- und Antworttupel an die horizontalen und vertikalen Axone Z_i und S_i (s. Abb. 22) zu legen und danach die Schreibleitung „Write (WR)“ (s. Abb. 22, Pin 17 (links oben)) auf Eins zu setzen. Damit das darauf folgende Frage-/Antworttupel nicht in diesen Lernvorgang „hineinrauscht“, muss die Schreibleitung „Write (WR)“ rechtzeitig mit Taktende dieses einen Taktes wieder auf Null fallen.

Zum Abfragen der Assoziativmatrix wird ein Schieberegister benutzt (s. Abb. 24), welches ausgehend von der obersten Zeile bei jedem Systemtakt über die Ausgänge Q_i die jeweils nachfolgende Matrixzeile auswählt. Ist für dieselbe Matrixzeile auch im Fragetupel eine Eins gesetzt, wird die „Lies (RD)“-Leitung aktiv (die horizontalen Axone Z_i in Abb. 22) und etwaige synaptische Verbindungen (Flip-Flops), die den Wert Eins gespeichert haben, legen diesen auf die „Ergebnis (OUT)“-Leitungen. Am „CLK“-Eingang liegt der Systemtakt, das Schieberegister wird durch eine Eins am „ST“-Eingang gestartet. Der „Z“-Ausgang meldet dem nachfolgenden Matrixabschnitt, dass die Abfrage dieses Matrixabschnitts beendet wurde; der „Z“-Ausgang wird also mit dem „ST“-Eingang des nachfolgenden Schieberegisters verbunden.

Der Zeitaufwand für das Abfragen der Matrix liegt in dieser Hardwarelösung in der Größenordnung O(n). Setzte man einen Integrator ein, einen Kondensator mit einer spannungsgesteuerten Stromquelle,[66] käme man womöglich sogar beim

[65] Man vergleiche demgegenüber das Eintragen eines Musters ins PAN-System (s. Kap. 3.2).
[66] vgl. [Tietze/Schenk 02, S. 570]

Abfragen der Matrix in eine Größenordnung O(1). Der genutzte Digitalsimulator erlaubte den Test eines so eingesetzten Operationsverstärkers allerdings nicht. Zudem lässt die Analogtechnik Ungenauigkeiten erwarten.

Bei einer Durchlauf-Verzögerung von etwa 1 ns bei einem Flip-Flop könnte man folglich in der Theorie in etwa 1 ns ein Muster lernen oder ein Muster abfragen, also etwa 10^9 Lern- oder Abfragevorgänge pro Sekunde vornehmen. Damit wären die Assoziativmatrizen unabhängig von ihrer Größe mit bis etwa 1 GHz Systemtakt betreibbar.

4.4 Schwellwertlogik

Als Schwellenwert θ erkennt die Schwellwertlogik der VIDAS -Maschine automatisch die Anzahl derjenigen horizontalen Axone, die beim Abfragen den Wert „1“ tragen.[67]

Von den „Ergebnis (OUT)“-Leitungen der Assoziativmatrix erhält eine Reihe von Digitalzählern (in Abb. 25 oben) Zählimpulse.[68] [69] Der Zähler ganz links zählt die Einsen im Fragetupel. Der Zählerstand dieses Zählers heiße Z_F. Darunter ordnen sich mehrere Reihen Komparatoren an, die die zu den „Ergebnis (OUT)“-Leitungen gehörenden Zählerstände mit dem von Z_F, $Z_F - 1$, $Z_F - 2$, $Z_F - 3$ vergleichen. An die Ausgabeleitungen der Schwellwertlogik gelangen die Ergebnisse der obersten Reihe, in der ein Komparator eine Gleichheit meldet; die Ergebnisse der darunter liegenden Reihen bleiben dann unberücksichtigt.

Diese Anordnung erlaubt es, eine nicht maximale Schwellwertgüte in mehreren Stufen anzuzeigen, was über die Ausgänge „-1“, „-2“, „-3“ und „-4“ geschieht.

Die Schwellwertlogik ist wie die Matrizen durch Zusammenstecken beliebig erweiterbar,[70] um auf die benötigte Breite der abzufragenden Matrix zu kommen. Zusätzlich könnten weitere Reihen für $Z_F - 4$, $Z_F - 5$ usw. in einfacher Weise ergänzt werden, falls die vierstufige Auffächerung zur Schwellwertgüte in einem Anwendungsfall einmal nicht ausreichen sollte.

[67] Das entspricht also der Anzahl der Einsen im Fragetupel.

[68] Vom PAN-System (s. Kap. 3.2) unterscheidet sich hier die Zählrichtung; beim PAN-System wird herunter-, hier wird heraufgezählt.

[69] Eine größere Abbildung befindet sich im Anhang in Abb. 87.

[70] vgl. Kap. 4.2

Abb. 25: Schwellwertlogik

4.5 VIDAS 1 und 2: Veranschaulichung der Vorgänge in einer Matrix

VIDAS 1 entstand Mitte Juli 1989, als sich ein Fernsehteam bei der Arbeitsgruppe zu Neuronalen Netzen an der Universität Osnabrück anmeldete,[71] um die Vorgänge in einer Assoziativmatrix beim Lernen und Suchen anschaulich und „filmbar“ vorgestellt zu bekommen.[72]

VIDAS 1 und 2 [73] dienten auch zur Veranschaulichung der Eingangskodierung ei-

[71] für die Wissenschaftssendung „Prisma“, NDR

[72] Das Wort VidAs entstand aus diesem Grund aus „videre“ und „associare“ und stellt selbstredend auch ein Wortspiel dar.

[73] VIDAS 1 wurde auf einem PC-XT mit 4,77 MHz als Programm in Turbo-Pascal entwickelt.

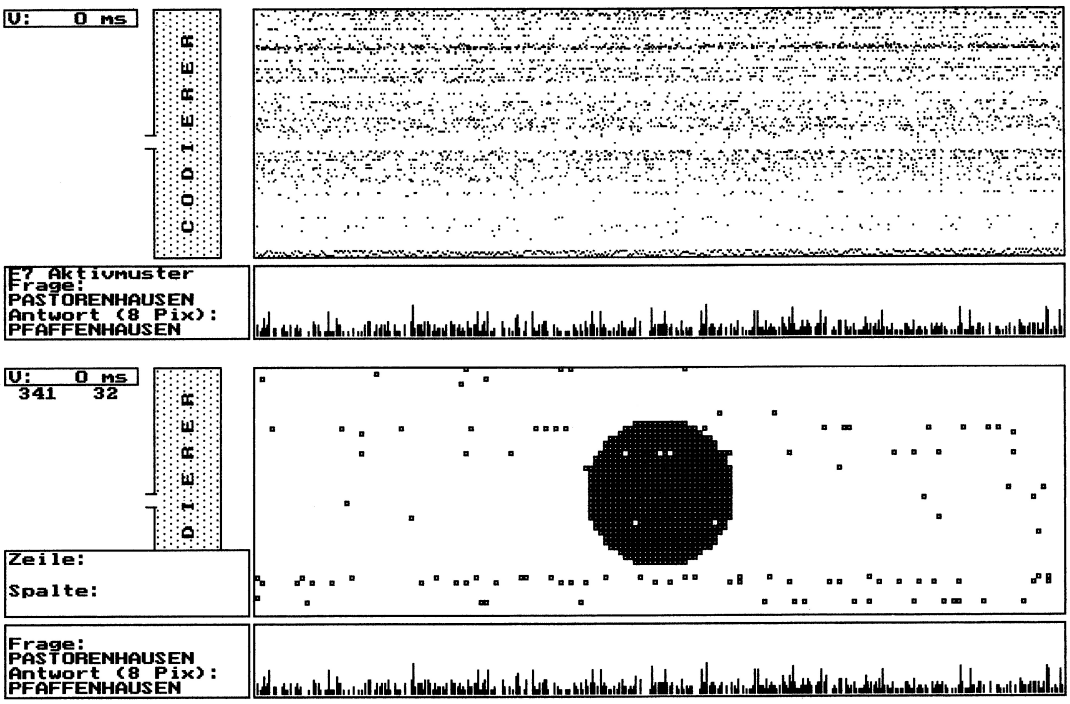


Abb. 26: VIDAS 1

ner Assoziativmatrix.[74] In diesem Zusammenhang wurde dann gezeigt, wie wenig anfällig eine Assoziativmatrix bei spärlicher Kodierung auf Zerstörungen reagiert (s. Abb. 26, untere Hälfte).[75] Dazu konnte man sich beliebige Bereiche der Matrix anzeigen lassen und per Mausklick synaptische Verbindungen verändern. Die mehreren Hundert Ortsnamen, die von der Matrix gelernt wurden, ließen sich dennoch fast alle finden, wie das Schwellwertbild (s. Abb. 26 ganz unten) verdeutlichte, welches sich in den auftretenden maximalen Schwellwerten von dem Schwellwertbild der ungestörten Matrix kaum unterschied. An der Universität Hildesheim wurde 1993 von Jan-Peter WILHELMS in „Datenträgerformatierung für SpaCAM“ diese Methode für den Test der Robustheit eines Speichers aufgegriffen und eine Diskette untersucht, auf der die Daten wie in einer Assoziativmatrix gespeichert wurden. Hinsichtlich der Auslesegüte verhielt sie sich gegenüber den experimentell herbeigeführten Defekten wie erwartet robust.

In dieser Zeit tauchten auch Probleme auf, in denen man sich mit Hilfe von Assoziativmatrizen fehlertolerant durch Entscheidungsbäume oder Graphen bewegen

VIDAS 2 war eine Portierung in Modula-2 auf einen etwa doppelt so schnellen Atari ST 1040.

[74] Doppelzeichen-Kodierung, s. Kap. 8

[75] zur spärlichen Kodierung s. z.B. [Bentz 89] oder [Hagström 96]

musste,[76] mit der Randbedingung, dass der Startknoten nicht gegeben war, sondern ebenfalls fehlertolerant und schnell aufgefunden werden sollte. Dazu waren Knoten mit Folgeknoten zu assoziieren. Die Resultate dieser Projekte bestärkten die Idee, einen Programmschritt mit seinem Nachfolger assoziieren zu wollen. PALM schreibt in „Cell Assemblies as a Guideline for Brain Research“[77] davon, dass man sich den Fluss des Bewusstseins als eindimensionalen Prozess vorstelle, der sich von „Punkt“ zu „Punkt“ oder von „Gedanke“ zu „Gedanke“ bewege.[78] In diesem Sinne veränderte das Projekt VIDAS seine Ausrichtung und untersuchte die Programmierbarkeit von Matrizenbündeln.

4.6 VIDAS 3: Störunanfälligkeit der Datenspeicherung

Vor dem Aufbau einer umfangreicheren Hardwarelösung mit den Ergebnissen aus Kapitel 4.1 entstanden mit VIDAS 3 und VIDAS 4 zwei Testumgebungen zur Erprobung der Störunanfälligkeit der aufzubauenden Matrizen in Software.

Nachdem Andreas JÜTTING 1990 in seiner Arbeit „Auslesegüte defekter Assoziativspeicher“[79] mit Mitteln der Stochastik die Fehlertoleranz von Assoziativmatrizen theoretisch untersucht hatte, war es das Ziel von VIDAS 3, mit Daten aus der Praxis im Experiment Antworten auf die Frage zu gewinnen, an wie viele gelernte Daten sich eine Assoziativmatrix trotz zunehmender Zerstörungen eindeutig „erinnert“. Es ergab sich augenfällig, wie und dass der Verlauf der „Vergessenskurve“ von der gewählten Zerstörungsart abhängt.

4.6.1 Ziele und Vorgehen

Mit Hilfe zufälliger Zerstörung der synaptischen Verbindungen einer bezüglich der Daten spärlich kodierten Assoziativmatrix[80] sollte deren Assoziationsfähigkeit hinsichtlich des korrekten Memorierens gelernter Zeichenketten veranschaulicht werden. In einem weiteren Schritt ging es um erste Erfahrungen bezüglich des korrekten Abarbeitens von Programmen (Sequenzen, Auswahlen, Wiederholungen). Da im Speicher eines Rechners seit VON NEUMANN nicht nur Daten, sondern auch Programme abgelegt werden,[81] sollte des Weiteren gezeigt werden,

[76] Projekte *BAUMAS* und *FOLGAS*

[77] s. [Palm 90, S. 137]

[78] „The flow of consciousness is described as a one-dimensional process which moves sequentially from 'point' to 'point' or from 'thought' to 'thought'.“ (aus [Palm 90, S. 137])

[79] vgl. [Jütting 90]

[80] „spärlich kodiert“ im Sinne von z.B. [Bentz 89]

[81] vgl. den Schluss von Kap. 3.2

dass auch Algorithmen in einer Assoziativmatrix in weiten Bereich gegen Störungen unanfällig ablaufen können.

Die Anfälligkeit herkömmlicher Datenträger gegen Zerstörung (head crash) oder wegen „Kippens“ einzelner Bytes (weak bytes) zeigt sich in der täglichen Praxis auch in den Regelungen zur Datensicherung in den Betrieben. Eine Ablage von Daten in einer Assoziativmatrix erweist sich demgegenüber in weiten Bereichen unempfindlich, wie VIDAS 3 verdeutlicht.

4.6.2 Beschreibung des Programms VIDAS 3

VIDAS 3 wurde unter Delphi 3.0 (FuLP-Version)[82] der Firma Borland entwickelt. Zur zufälligen Zerstörung der synaptischen Verbindungen einer Assoziativmatrix wurde der in Delphi 3.0 implementierte Pseudozufallszahlengenerator für gleichverteilte Zufallszahlen benutzt.

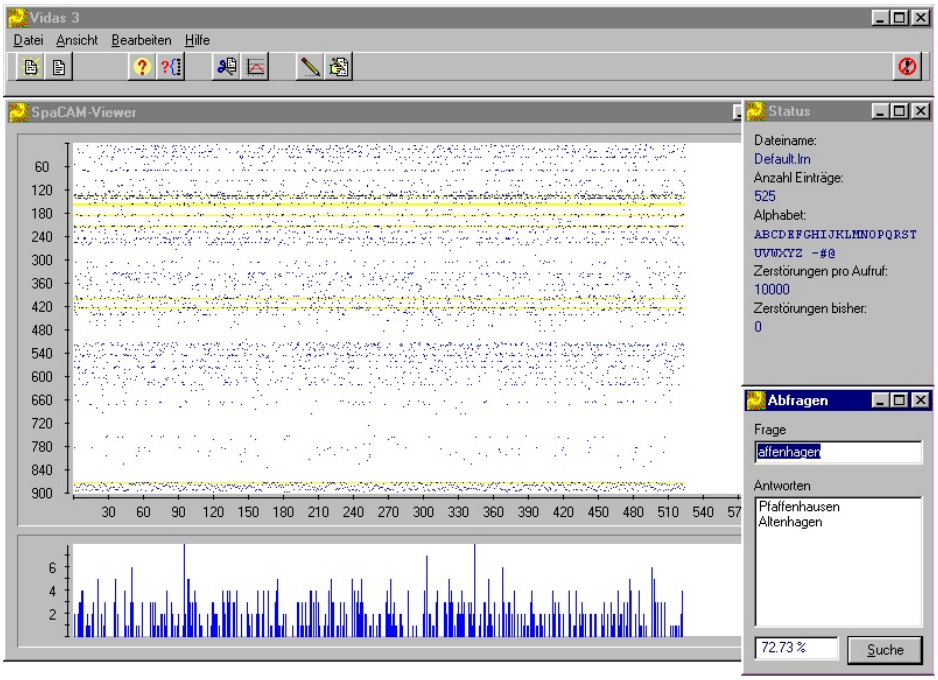


Abb. 27: Abfrage an eine Assoziativmatrix mit 540.000 Synapsen

Nach Aufruf von VIDAS 3 erscheinen auf dem Bildschirm vier Objekte (s. Abb. 27): die Steuerleiste (oben), das Statusbord (rechts oben), die Abfragebox (rechts unten) und ein Fenster mit der Darstellung der aktuellen Assoziativmatrix (links oben) und des aktuellen Schwellvektors (links unten).

[82] FuLP — Forschung und Lehre-Programm

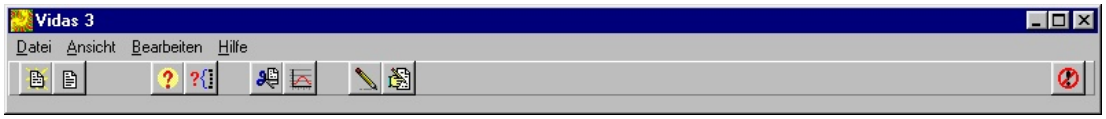


Abb. 28: Steuerleiste von VIDAS 3

Über die Steuerleiste (s. Abb. 28) erhält man durch eine Menüzeile oben mit den Einträgen Datei, Ansicht, Bearbeiten und Hilfe und eine Reihe von Schaltflächen (angereiht auf einer Leiste unten) Zugang zum Funktionsumfang von VIDAS 3. Den Schaltflächen (Buttons) sind (von links nach rechts aufgezählt) folgende Funktionen zugeordnet:

- Lerndatei öffnen (lädt eine Datei mit zu lernenden Begriffen)
- Programmdatei öffnen (lädt eine Programmdatei in die Assoziativmatrix)
- Abfragen (aktiviert die Dialogbox zum Abfragen von Begriffen)
- Selbsttest (testet die Assoziativmatrix mit allen gelernten Begriffen)
- Zerstörungen (zufälliges Ändern der synaptischen Verbindungen der Assoziativmatrix)
- Zerstörstatistik (grafische Darstellung andauernder Zerstörungen)
- Lerndatei-Editor (erlaubt das Ändern (und Neulernen) einer Lerndatei)
- Programmdatei-Editor (erlaubt das Verfassen von VIDAS -Programmen)
- VIDAS 3 beenden

Dem Statusbord (s. Abb. 29) kann man Informationen zum Zustand der aktuellen Assoziativmatrix entnehmen. Neben dem Dateinamen der Lern- oder Programmdatei ist es die Anzahl der gelernten Assoziationspaare (in Abb. 29 sind es 525), das Alphabet über dem kodiert wurde, wie viele Zerstörungen pro Aufruf vorgenommen werden und wie viele Zerstörungen bisher bereits vorgenommen wurden (hier: 20.000).

Im Eingabefeld (in Abb. 30 unterhalb der Beschriftung mit 'Frage') trägt man den zu suchenden Begriff ein (hier: 'Affenhagen') und setzt durch Betätigen der Taste Enter oder Anklicken des Buttons 'Suche' oder mit Alt-S fort. Anschließend finden sich in der Listbox (unterhalb der Beschriftung mit 'Antworten') die mit gleichem maximalen Schwellwert assoziierten Ortsnamen (hier: 'Pfaffenhausen' und 'Altenhagen'). Neben dem Button 'Suche' wird die 'Güte' der Ähnlichkeitssuche (hier 72,73 %) ausgegeben.

Im oberen Teil des Assoziativmatrix-Viewers (s. Abb. 31) wird die Verteilung der gesetzten Bits der Assoziativmatrix dargestellt. In der Vertikalen ist die Darstellung gestaucht, man kann aber durch einen Doppelklick in dieses Fenster eine

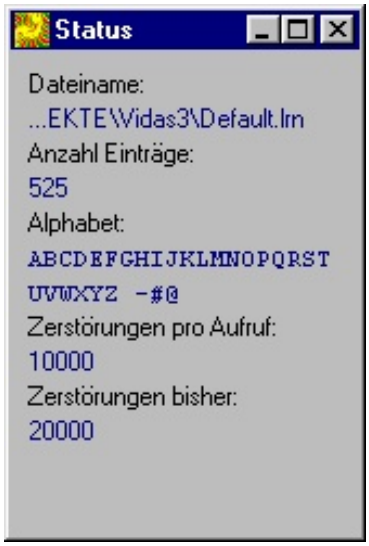


Abb. 29: Statusbord mit Daten zur aktuellen Sitzung von VidAs 3

vergrößerte Ansicht abrufen. Nach einer Abfrage sind diejenigen Zeilen der Assoziativmatrix gelb markiert, die während der Abfrage aktiviert wurden (Aktivitätsmuster).

Im unteren Teil des Viewers werden die zur letzten Abfrage gehörende Schwellwerte wiedergegeben. Durch einen Doppelklick in dieses Fenster erreicht man eine vergrößerte Darstellung und kann sich dabei z.B. auch nächstähnliche Begriffe anzeigen lassen (durch Bewegen der Maus im Vergrößerungsfenster).

4.6.3 Zerstörungsarten

Nach Betätigen der entsprechenden Schaltfläche in der Steuerleiste kann man zwischen vier Arten von zufälligen Zerstörungen wählen:

- Gießkanne (zufälliges Löschen von Bits)
- Lochen (kreisförmiges Löschen von Bits)
- Kreis setzen (kreisförmiges Setzen von Bits)
- Kreis XOR (kreisförmiges Toggeln von Bits)

Dem Statusbord (s. Abb. 29) kann man die Anzahl der bereits getätigten Zerstörungen entnehmen. Nach den Zerstöraktionen wird man entweder gezielt über die Abfragedialogbox nach noch erkannten Begriffen fragen oder alle Begriffe durch den Selbsttest (s. Abb. 32) abprüfen wollen.

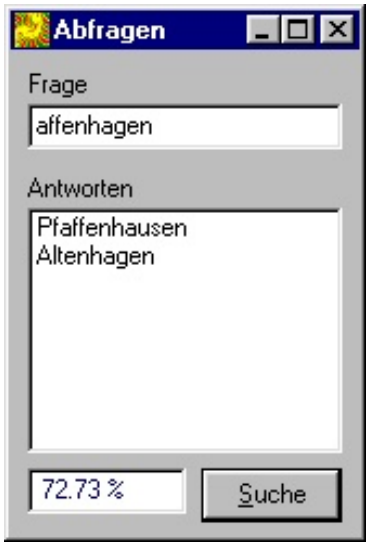


Abb. 30: Abfragedialogbox von VIDAS 3

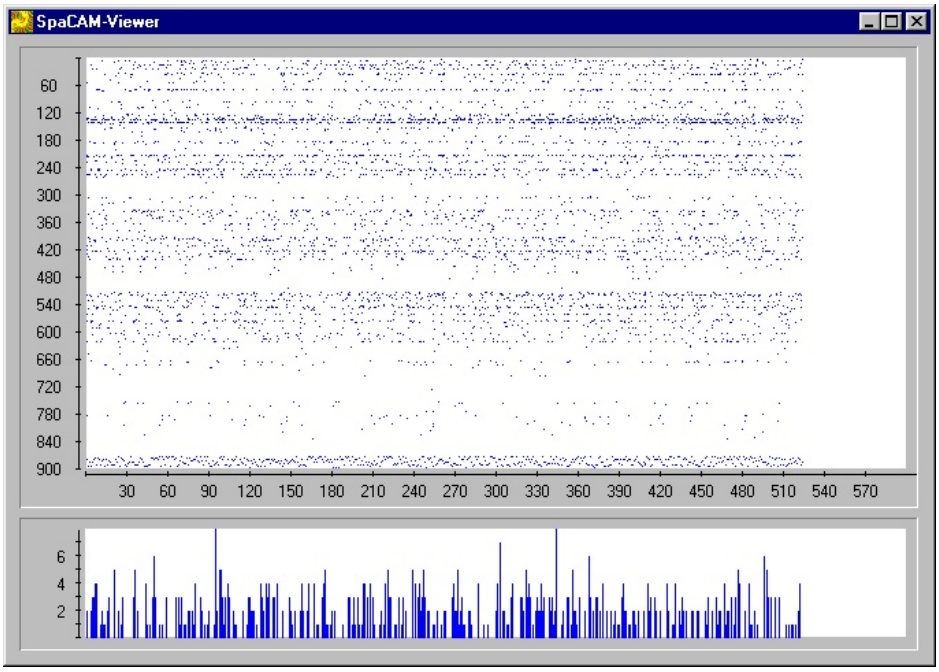


Abb. 31: Assoziativmatrix-Darstellung von VIDAS 3

Nach dem Start des Selbsttests (s. Abb. 32) wird ausgegeben, wie viele Ortsnamen noch erkannt wurden (hier nach 20.000 Zerstörungen noch 522 von 525).

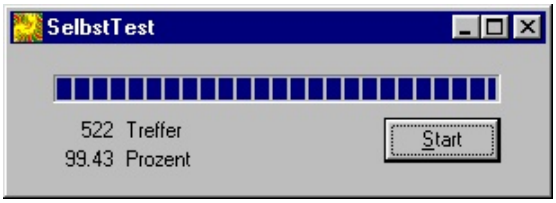


Abb. 32: Selbsttest von VIDAS 3

Die Schaltfläche zum 'Dauerzerstören' bewirkt die Anzeige einer Dialogbox, in der man vor dem Start die Anzahl der Zerstörungsdurchläufe angeben kann. Abbildungen dazu finden sich im Kapitel 4.6.7 'Ergebnisse'.

4.6.4 Lerndateien

Die Einträge innerhalb einer Lerndatei kann man anzeigen und verändern, indem man den entsprechenden Button (s. Abb. 28) anwählt. Der Editor verhält sich in seiner Bedienung wie ein gewöhnlicher Windows-Texteditor. Entweder trägt man pro Zeile nur jeweils einen Begriff ein oder – durch einen Backslash \ voneinander getrennt – ein Assoziationspaar (z.B. Haus\Garten, wenn bei Abfrage mit dem Begriff 'Haus' als Antwort der Assoziativmatrix 'Garten' erfolgen soll.

Im Fenster des Lerndateieditors findet man oben eine Menüzeile, über deren Menüpunkt 'LernDatei' ein zwischenzeitliches Abspeichern, Löschen, Öffnen usw. bewirkt werden kann.

4.6.5 VIDAS 3-Programme

Zum Schreiben von VIDAS 3-Programmen dient ein besonderer Programmeditor (s. Abb. 34), der sich in zwei Teilen darstellt. Im linken Fenster (in heller-er/rötlicher Schrift) werden Programmzeilen'nummern' (Zeilenschlüssel) generiert, die der Anwender zwar auch von Hand eintragen oder ändern könnte, dies aber auch dem Editor überlassen kann, weil er gegebenenfalls fehlende 'Nummern' automatisch erzeugt.

Im rechten Fenster des Programmeditors wird die gewünschte Abfolge an Aktionen eingegeben, die die Assoziativmatrix lernen soll. Zur Unterstützung des Programmierenden erhält dieser nach Doppelklick mit der Maus in dieses Fenster ein PopUp-Menü, aus welchem er die möglichen Befehle wählen kann. Sich anschließende Dialogboxen versorgen die Befehle mit den etwaigen Parametern.

Abb. 33: Editor für Lerndateien bei VIDAS 3

Modell für die Programmierumgebung ist eine Turtlegrafik-Umgebung wie man sie beispielsweise von LOGO her kennt.[83] Die Turtle wird dabei über eine Zeichenfläche geschickt, auf der sie ihre Spuren hinterlässt. Die hier vorgestellte Turtle kann zudem 'sprechen', wenn der Rechner über eine Soundkarte verfügt. Ferner ist der Turtle eine zweite Assoziativmatrix zugeordnet, die als 'Kurzzeitgedächtnis' fungiert.

Als Befehle wurden in VIDAS 3 folgende implementiert:

- Gehe {vorwärts | rückwärts}
- Drehe dich nach {links | rechts}
- Schreibe <AssoTerm>
- Sprich <AssoTerm>

[83] s. z.B. [Abelson 83]

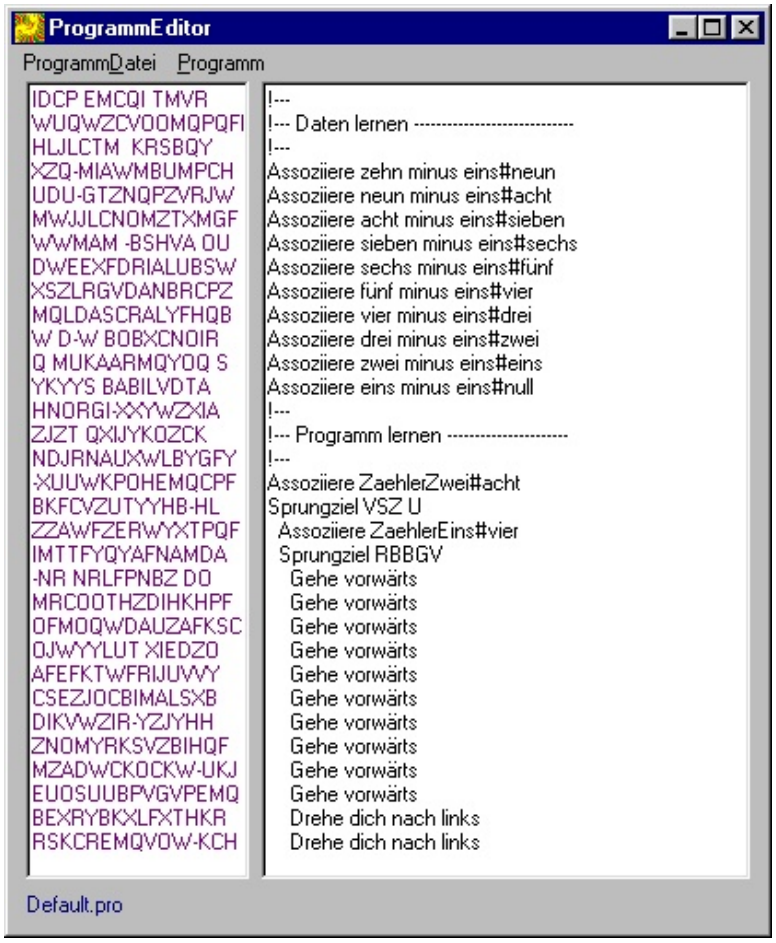


Abb. 34: Editor für Programmdateien bei VidAs 3

- Assoziiere <Bezeichner>#<AssoTerm>
- Sprungziel <Sprungziel>
- Springe –> <Sprungziel>
- Springe falls Null <Bezeichner> –> <Sprungziel>

In einem <AssoTerm> werden alle Teilterme, die in Klammern stehen, durch mittels des Kurzzeitgedächtnisses assoziierte Begriffe ersetzt.

Im Fenster des Programmdateieditors findet man oben eine Menüzeile, über deren Menüpunkt 'ProgrammDatei' ein zwischenzeitliches Abspeichern, Löschen, Öffnen usw. erfolgen kann. Der zweite Menüpunkt 'Programm' erlaubt den Test (Syntax) und den Start des angezeigten Programms.

4.6.6 Laufende Programme

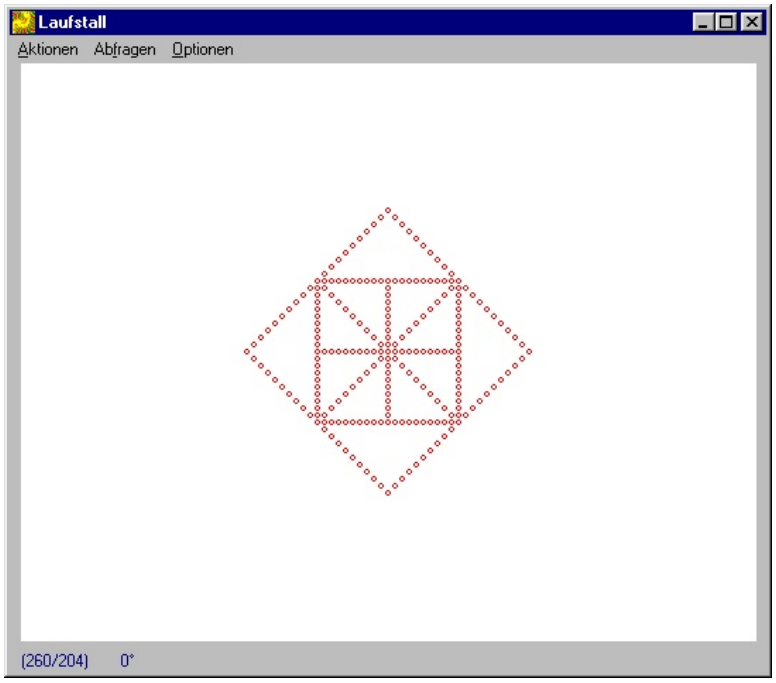


Abb. 35: Fenster für das ablaufende Programm bei VidAs 3

Die Aktionen der Turtle werden in einem eigenen Fenster (s. Abb. 35) mit einer Zeichenfläche in seiner Mitte wiedergegeben. Über eine Menüzeile kann die Turtle zurückgesetzt und das Programm neu gestartet werden. Außerdem ist das Abfragen der Assoziativmatrix für das Kurzzeitgedächtnis möglich.

Möchte man mit Zerstörungen der Programm - Assoziativmatrix experimentieren, empfiehlt es sich, über den Menüpunkt 'Optionen' (s. Abb. 35) gegebenenfalls die maximale Anzahl der abzuarbeitenden Aktionen in diesem Fenster zu begrenzen, da die Turtle durch die Zerstörungen in endlose Wiederholungen gleicher Aktionen geraten kann.

Im unteren Rand des Fensters wird die aktuelle Position und Orientierung der Turtle angezeigt. Ferner tauchen hier unten etwaige Fehlermeldungen auf (unbekannte Aktionen, die die Turtle ausführen soll).

4.6.7 Ergebnisse von VidAs 3

Folgende Diagramme zeigen die Anzahlen der erkannten Begriffe (in Prozent) gegenüber den Anzahlen der Zerstörungen (10^4 Zerstörungen pro Skaleneinheit).

Werden pro Zerstörungsaktion der Art 'Gießkanne' (s. Abb. 36) zufällig 10^4 Bits der Matrix angefahren und gelöscht, so erbrachten die Beobachtungen stets einen Verlauf, der einer Hälfte einer Gaußschen Glockenkurve ähnelt. Bei etwa 40.000 Aktionen sank die Erkennungsleistung der Assoziativmatrix auf 75 %, bei etwa 70.000 Aktionen auf die Hälfte und nach etwa 140.000 Aktionen auf 25 %.

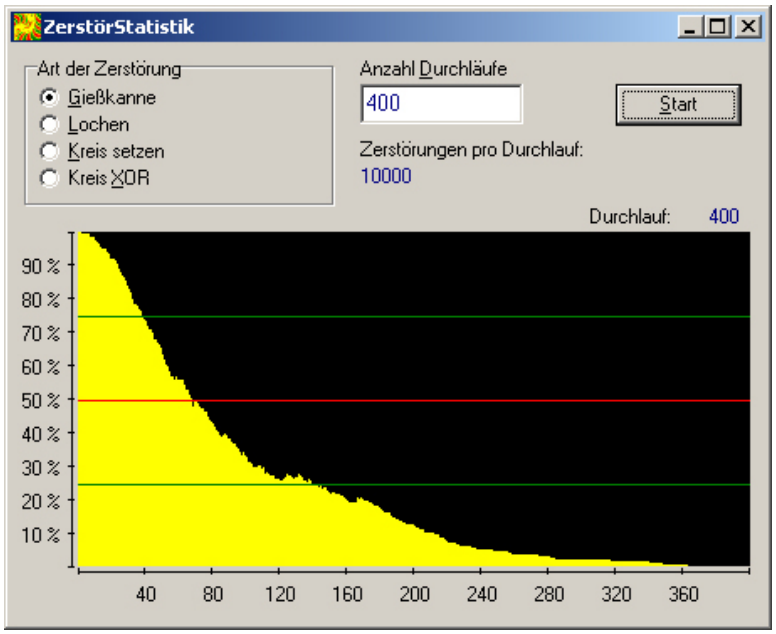


Abb. 36: Zufälliges Löschen von Bits der Assoziativmatrix

'Locht' man die Assoziativmatrix (s. Abb. 37), das heißt, die 10^4 Zerstörungen werden in Kreisen mit einem Durchmesser von je 30 Bits vorgenommen, so erhielt man einen von der 'Gießkannen'-Zerstörung kaum abweichenden Verlauf. Nach etwa 40.000 Löschaktionen sank die Anzahl korrekt erinnerter Zeichenketten auf 75 %, nach etwa 70.000 Aktionen auf die Hälfte und nach ungefähr 120.000 Löschungen auf ein Viertel.

Setzt man (statt zu löschen) neue Bits kreisförmig (wie beim Lochen) in die Assoziativmatrix (s. Abb. 38), so ist unter den assoziierten Zeichenketten immer auch die gelernte. Allerdings diskriminiert die Assoziativmatrix immer schlechter zwischen den gelernten Begriffen, was hier nicht dargestellt wurde.

Zerstört man die Assoziativmatrix indem man ihre Bits kreisförmig (wie beim Lochen) toggelt (logisch negiert), so beobachtete man immer einen gegenüber den oben genannten Zerstöraktionen verschiedenen Verlauf der Erkennungsleistung (s. Abb. 39). Bei etwa 15.000 Zerstörungen dieser Art wurden hier nur noch 75 % der

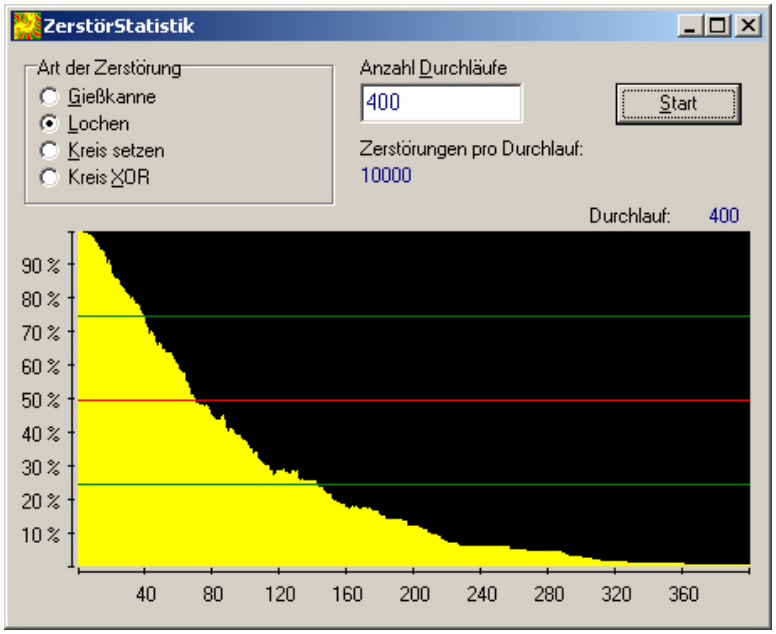


Abb. 37: Zufälliges kreisförmiges Löschen von Bits der Assoziativmatrix

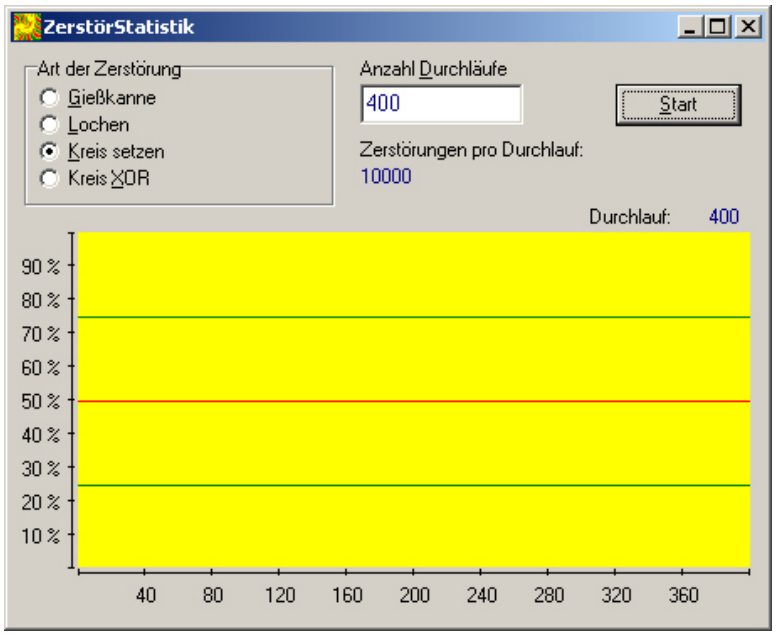


Abb. 38: Zufälliges kreisförmiges Setzen von Bits der Assoziativmatrix

Zeichenketten erkannt, bei etwa 20.000 waren es die Hälfte und bei etwa 30.000 ein Viertel.

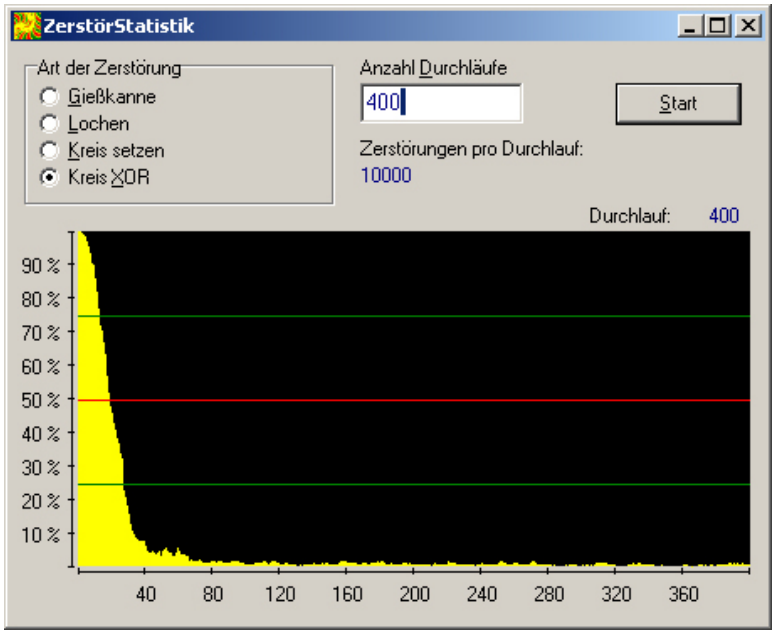


Abb. 39: Zufälliges kreisförmiges *Toggeln* von Bits der Assoziativmatrix

4.7 VIDAS 4: Störunanfälligkeit beim Abarbeiten von Programmen

In VIDAS 4 wurde wie in VIDAS 3 der Frage nachgegangen, welchen Einfluss Zerstörungen haben, wenn man in einer Assoziativmatrix Anweisungen für die Bewegung einer Turtle ablegt. Allerdings wurde in VIDAS 4 das fehlertolerante Vorgehen bei bedingten Sprüngen, Wertzuweisungen und Speicherabfragen besser umgesetzt und beobachtet, um Konstruktionsmängel des unten beschriebenen VIDAS 4-Modells aufzuspüren. Im Folgenden seien auch die Entwurfschritte für diese drei Beobachtungsschwerpunkte wiedergegeben, da sich aus ihnen viele Entscheidungen für die VIDAS -Maschine in Kap. 4.9 ergeben haben.

4.7.1 Matrizen als Programmspeicher

Durch das Projekt VIDAS 3 wurde eine Testumgebung entwickelt, in der man den fehlertoleranten Ablauf von Folgen von Aktionen bezüglich seiner Störunanfälligkeit beobachten konnte.[84]. Dazu wurden zwei Assoziativmatrizen eingesetzt: die erste assoziierte zu einer gegebenen Situation (Programmzeile) die nachfolgende und ei-

[84] vgl. „VIDAS 3 - Untersuchung zerstörter Assoziativmatrizen“ vom 8.5.98; vgl. Kap. 4.6

ne zweite Matrix diente als Variablenspeicher („Kurzzeitgedächtnis“).[85] Die Interpretation der zur Situation gehörenden Programmzeilen (zum Beispiel bedingte Sprünge oder Wertzuweisungen) übernahm in VIDAS 3 die (nicht-fehlertolerante) Testumgebung. VIDAS 4 sollte demgegenüber zeigen, dass mit den Matrizen zur Laufzeit auch Wertzuweisungen, Speicherabfragen und bedingte Sprünge verwirklicht werden können, so dass keinerlei (nicht-fehlertolerante) Technik für das Abarbeiten von Programmen durch eine spätere VIDAS -Maschine benötigt wird.

In der Testumgebung von VIDAS 3 erhielt jede Programmzeile als Zeilennummer eine selbstgenerierte Zufallszeichenkette (s. Abb. 34, linkes Fenster im Programmeditor), über die eine Programmzeile mit ihrer Nachfolgerin assoziiert wurde. Die Interpretation und Ausführung der Programmzeilen übernahm die Testumgebung. Also wurde beispielsweise für die Zeile 'Assoziiere ZaehlerZwei#acht' herkömmlicher Programmcode aktiv, der in die Matrix für das „Kurzzeitgedächtnis“ dieses Frage-Antwort-Paar eintrug. Noch umfangreicher wurde der Einfluss herkömmlichen Programmcodes bei bedingten Sprüngen (s. 'Springe falls Null ZaehlerZwei->Sprungziel DF JY'). Zum Aufbau einer realistischen Testumgebung für einen späteren fehlertoleranten Prozessor ist diese Form der Implementation nicht geeignet.

4.7.2 Sequenzen

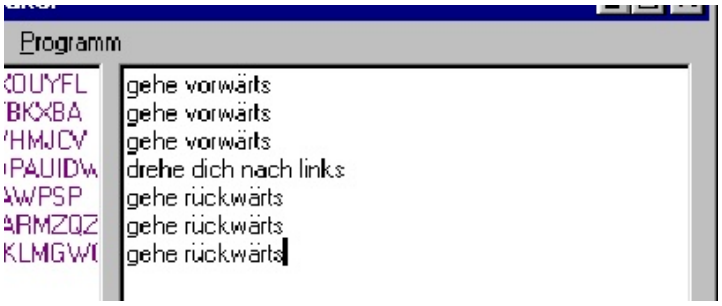


Abb. 40: Sequenzen in VIDAS 3

Hinsichtlich ihrer Komplexität erfordern einige Programmzeilen zu ihrer Abarbeitung eine relativ einfache Matrix-Umgebung, während andere (zum Beispiel für bedingte Sprünge und Speicherabfragen) eine aufwändigere Struktur bedingen. Programme einfacher Art haben etwa die Gestalt wie in Abbildung 40. Sie ließen sich durch eine einzige Matrix A (s. Abb. 41) fehlertolerant abarbeiten,

[85] Es wurde für den Variablenspeicher der Einsatz von RS-Flip-Flops wie in Kap. 4.1, Abb. 18, angenommen, nicht der von JK-Flip-Flops.

in welchem die Antwort von A ('Abfolge') als nächste Frage an A dient. Dieser Prozess $p1$ verantwortet den Ablauf des Programms und wird in seiner Hardware-Verwirklichung durch einen Taktgeber vorangestoßen. Die Matrix B ('Befehle') dieses einfachen Matrixverbundes assoziiert zur Frage an A Aktionen, von denen eine zum Beispiel 'gehe rückwärts' für eine LOGO-Turtle bedeuten könnte. Die Aktivierung der Ausgangsleitungen b kann wie ein Zug an einem der Fäden einer Marionette verstanden werden. Ein Dekodierer für die Ausgangsaktivierung würde in diesem einfachen Hardwaremodell Aktionen auf Displays, Relais, Lautsprechern usw. auslösen. Programmteile, über die Speicherabfragen oder Speichereinträge bewirkt werden sollen, lassen sich durch dieses erste einfache Modell nicht abarbeiten, da keine Eingangsbahnen e (in Umkehrung zu b) vorhanden sind.

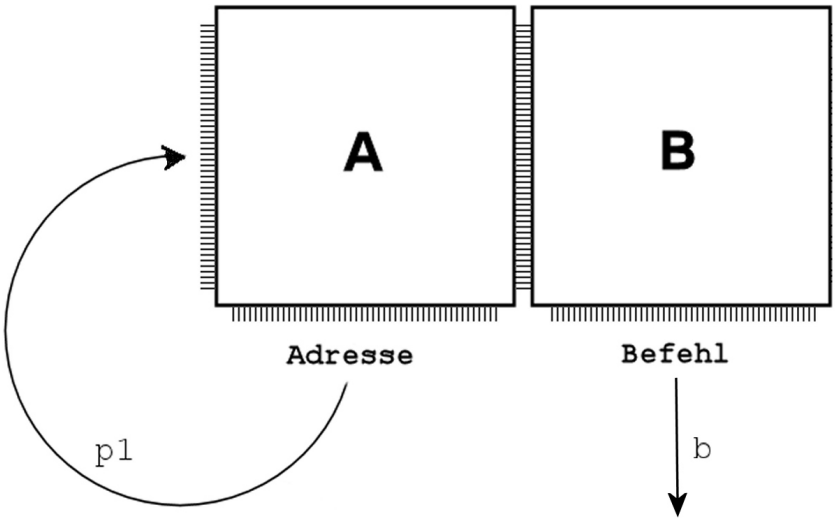


Abb. 41: Abfolge von Programmschritten bei VidAs 4

4.7.3 Wertzuweisung an Variable

Um zur Laufzeit des Programms etwas „lernen“ zu können, wird der Umgang mit Variablen ermöglicht, die ihre Werte erst zur Laufzeit durch eine Zuweisung erhalten.

Zur Lösung des Problems des Schreibens in den Speicher zur Laufzeit des Programms soll das einfache Modell aus Abb. 41 durch eine Matrix K („Kurzzeitgedächtnis“) und einen zweiten Prozess $p2$ ergänzt werden, welcher bei jedem Durchlauf von $p1$ angestoßen wird und bei Vorliegen einer auslösenden Aktivierung von

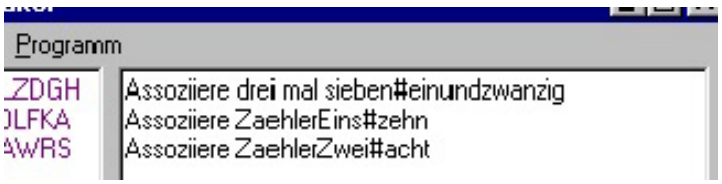


Abb. 42: Wertzuweisung an Variable bei VIDAS 4

b das ebenfalls über b gelieferte Frage-Antwort-Paar ($e1$; $e2$) in K einträgt (s. Abb. 43). $p1$ wird durch $p2$ an der Stelle $x1$ unterbrochen. Der Dekoder d legt die Ausgangsaktivierung von b nur dann an K an, wenn ein besonderer Teil von b den Schreibvorgang auf K freisetzt, was beispielsweise beim 'Assoziiere'-Befehl eines VIDAS 4-Programms wie in Abb. 42 passieren soll, wenn der Variablen 'drei mal sieben' der Wert 'einundzwanzig' zugeordnet wird (s. 'Assoziiere drei mal sieben#einundzwanzig'). $p2$ versucht den Schreibvorgang auf K in jedem Falle. Setzt der Dekoder dann aber keinen Schreibvorgang frei, so hinterlässt der Vorgang in K keine Veränderung.

4.7.4 Bedingte Sprünge

Das programmgesteuerte Auslesen des Matrix-Speichers, wie das u.a. bei bedingten Sprüngen nötig ist, bedarf weiterer Ergänzungen des bisher entwickelten VIDAS 4-Modells. Absolute Sprünge sind bereits mit dem ersten, einfachen Modul zu verwirklichen. Bedingte Sprünge sind jedoch Voraussetzung zum Abarbeiten beliebiger Algorithmen.

Dazu sei als Beispiel ein aus Assemblersprachen bekannter bedingter Sprungbefehl JNZ (z.B. 'JP NZ,pq', d.i. jump on not zero to location pq) gewählt.[86] Der betreffende von Neumann-Prozessor führt diesen Sprung aus, wenn in einem besonderen Prozessorregister ein Signal entsprechend gesetzt ist.

Im VIDAS 4-Modell soll ein solcher bedingter Sprung wie in Abb. 45 dargestellt abgearbeitet werden: am Ende des Prozesses $p2$ wird stets ein dritter Prozess $p3$ ausgelöst, der mit der Ausgangsaktivierung b die Matrix K abfragt, eine Aktivierung, die genau nur dann an K anliegt, wenn der Dekoder d als Aktion einen JNZ-Befehl erkannt hat. Die Falltür f gibt das Signal für A nur in diesem Falle frei, was $p3$ sicherzustellen hat. Da alle Prozesse bei jedem Prozessortakt unbedingt ausgeführt werden, wird f nötig, weil sonst die Ausgangsaktivität von $e3$ unkontrolliert Einfluss auf den Programmablauf nehmen kann.

[86] vgl. Rodnay Zaks: „Programming the Z80“, 2nd edition, S. 282f

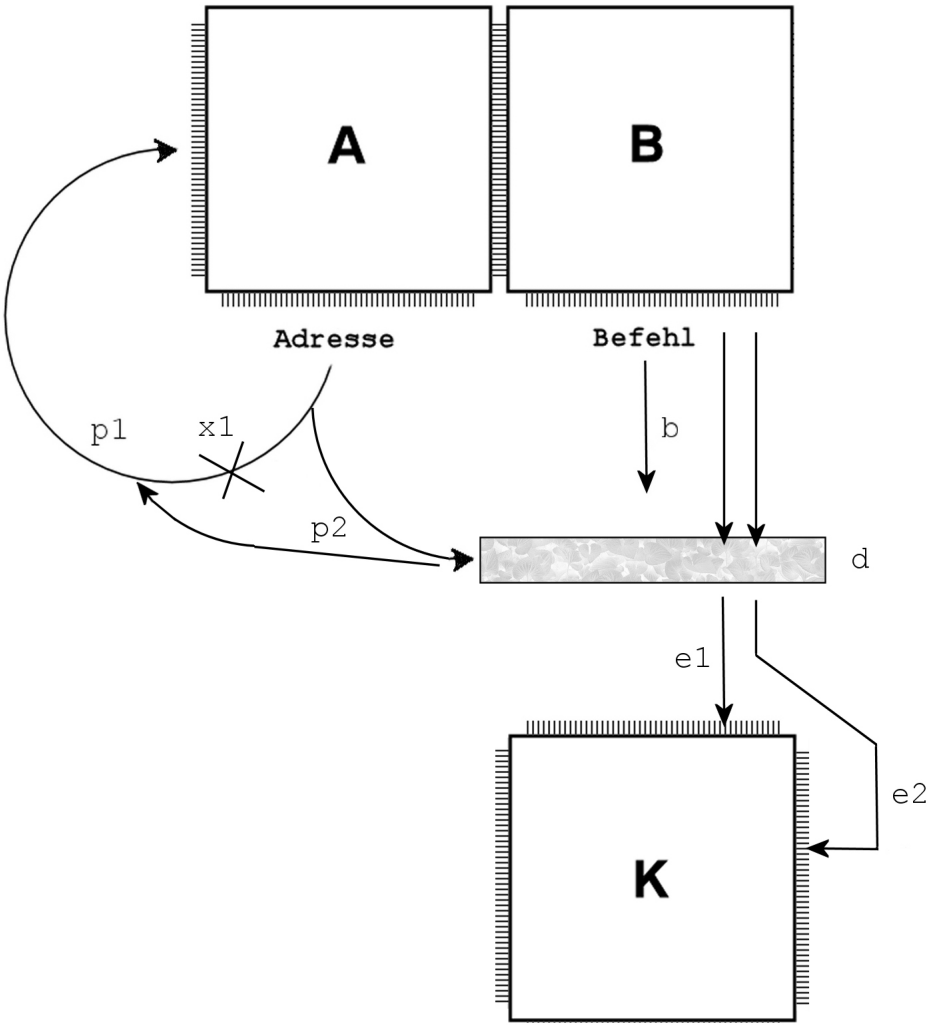


Abb. 43: VidAs 4-Modell mit Variablenspeicher

Erkennt d einen JNZ-Befehl, legt er die Ausgangsaktivierung b für $e1$ an. $p3$ veranlasst die Abfrage von K mit der $e1$-Aktivität, so dass an $e3$ die Antwort steht. Beim JNZ-Ereignis gibt $p3$ die Falltür f frei, so dass an A eine Frage anliegt, die sich aus zwei Anteilen zusammensetzt, dem Anteil der von A selbst im letzten Takt geantwortet wurde und dem von K stammenden Anteil. Damit jetzt das richtige Sprungziel (fehlertolerant) gefunden wird, nutzt man die Eigenschaft einer Assoziativmatrix aus, auf Anfragen mit mehr Einträgen im Fragetupel anders zu antworten als auf solche mit weniger Einträgen. Dazu speichert man zur Programmlernzeit zum einen das Frage-Antwort-Paar ab, welches beim

Abb. 44: Bedingte und absolute Sprünge in VidAs 4

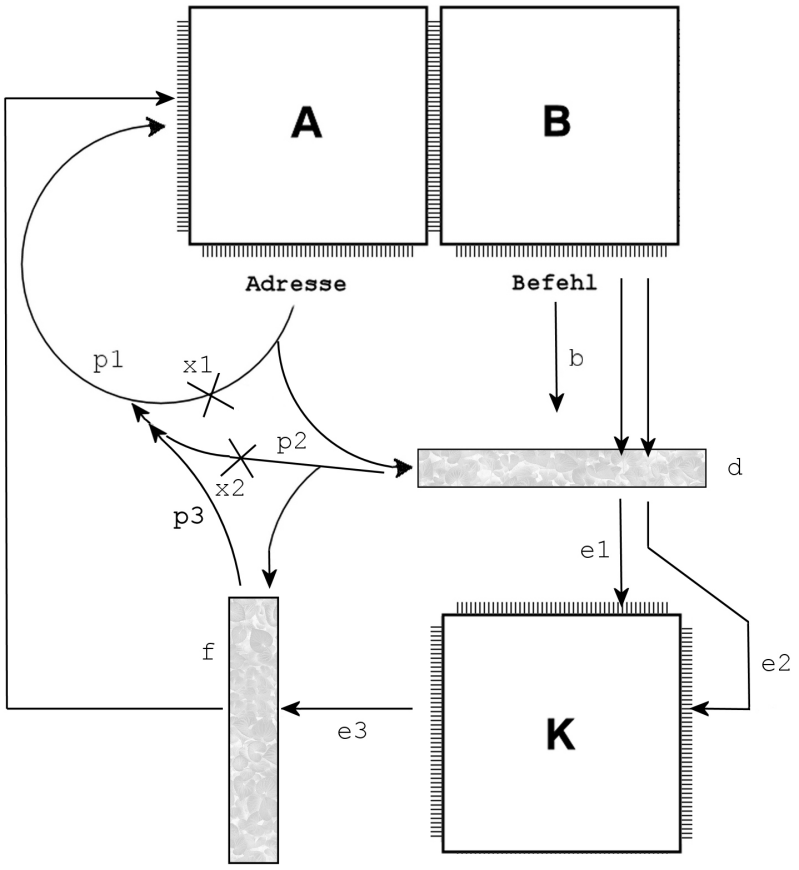


Abb. 45: VidAs 4-Modell mit bedingten Sprüngen

Nichteintreten des JNZ-Ereignisses wirksam werden soll und dann zusätzlich alle Frage-Antwort-Paare, die bei den verschiedenen Antworten von K zu anderen Sprungzielen führen sollen.

In der Programmiersprache von VidAs 4 werden bedingte Sprünge wie in Abb. 44 angegeben ('Springe falls Null ZaehlerEins->Sprungziel <marke>'). In diesem Beispiel wird der Programmablauf zur Programmzeile <marke> verzweigen, falls die Variable ZaehlerEins den Wert Null hat.

Das Kurzzeitgedächtnis K bedarf im VIDAS 4-Modell besonderer Aufmerksamkeit, da in ihm laufend Frage-Antwort-Paare abgelegt werden (insbesondere bei Schleifenkonstrukten), bei denen die Frage (Variablenname) oft die gleiche ist, die Antwort jedoch ständig wechselt. Entweder entscheidet man sich, immer die letzte Antwort aus der Liste aller Antworten zu wählen, oder man löscht vor jedem Schreibvorgang in den aktivierten Matrixzeilen und -spalten gezielt einige Bits.

4.7.5 Speicherabfrage zur Laufzeit

Zur Laufzeit müssen Werte von Variablen abgefragt und weiterverarbeitet werden können. Eine Programmzeile der Art 'Sprich [Gruss]' soll bewirken, dass zuerst der Wert der Variablen 'Gruss' aus dem Kurzzeitgedächtnis K geholt und anschließend dieser Wert an die Aktion 'Sprich' als Parameterwert weitergereicht wird. Die Programmzeile 'Addiere ZaehlerEins, [ZaehlerZwei]' wird abgearbeitet, indem der Wert der Variablen 'ZaehlerZwei' vermöge K ermittelt und dieser dann zum Wert der Variablen 'ZaehlerEins' addiert wird. Die Summe ist im letzten Schritt wieder mit 'ZaehlerEins' zu assoziieren und also in K als neuer Wert von 'ZaehlerEins' einzutragen. An diesen Beispielen wird deutlich, dass Programmzeilen mit Speicherabfragen nicht mehr nur in einem einzigen Schritt (Takt) erledigt werden können. In Assemblersprachen wäre die Vorgehensweise so, dass man Werte aus dem Speicher holt und in bestimmten Registern ablegt, in denen die dann in Folge aufzurufenden Routinen (hier z.B. 'Sprich') ihre Parameterwerte erwarten. Ein entsprechendes Vorgehen kann man mit dem VIDAS 4-Modell erreichen, wenn man $e3$ als Register für diese Parameterwerte versteht.

Programmzeilen wie 'Sprich [Gruss]' oder 'Addiere ZaehlerEins, [ZaehlerZwei]' zerlege man dazu in zwei Schritte, nämlich in a) '[Gruss]' und in b) 'Sprich' beziehungsweise in a) '[ZaehlerZwei]' und in b) 'Addiere ZaehlerEins'. Bei Abarbeitung der Anweisungen b) erwarten die zugehörigen Routinen ihre Parameterwerte in $e3$, also als Ausgangsaktivität von K, die folglich von den vorher ablaufenden Prozessen $p1, p2, p3$ nicht gelöscht worden sein darf. Anweisungen a) sollen im VIDAS 4-Modell ähnlich wie bedingte Sprünge erledigt werden, ohne dass die Falltür f freigegeben wird. Das heißt, der Dekoder d erkennt den Speicherabfragebefehl und legt $e1$ an K an. $p3$ veranlasst die Abfrage von K mit der $e1$-Aktivität, gibt f aber nicht frei. Folglich liegt der mit $e1$ assoziierte Wert auf $e3$.

4.7.6 Beschreibung des Programms VIDAS 4

Das VIDAS 4-Programm führt das oben skizzierte VIDAS 4-Modell aus. Das Programm wurde unter *Delphi 4.0* (FuLP-Version) der Firma Borland entwickelt.

Zur zufälligen Zerstörung der Bits einer Assoziativmatrix wurde der in *Delphi 4.0* implementierte Pseudozufallszahlengenerator für gleichverteilte Zufallszahlen benutzt.

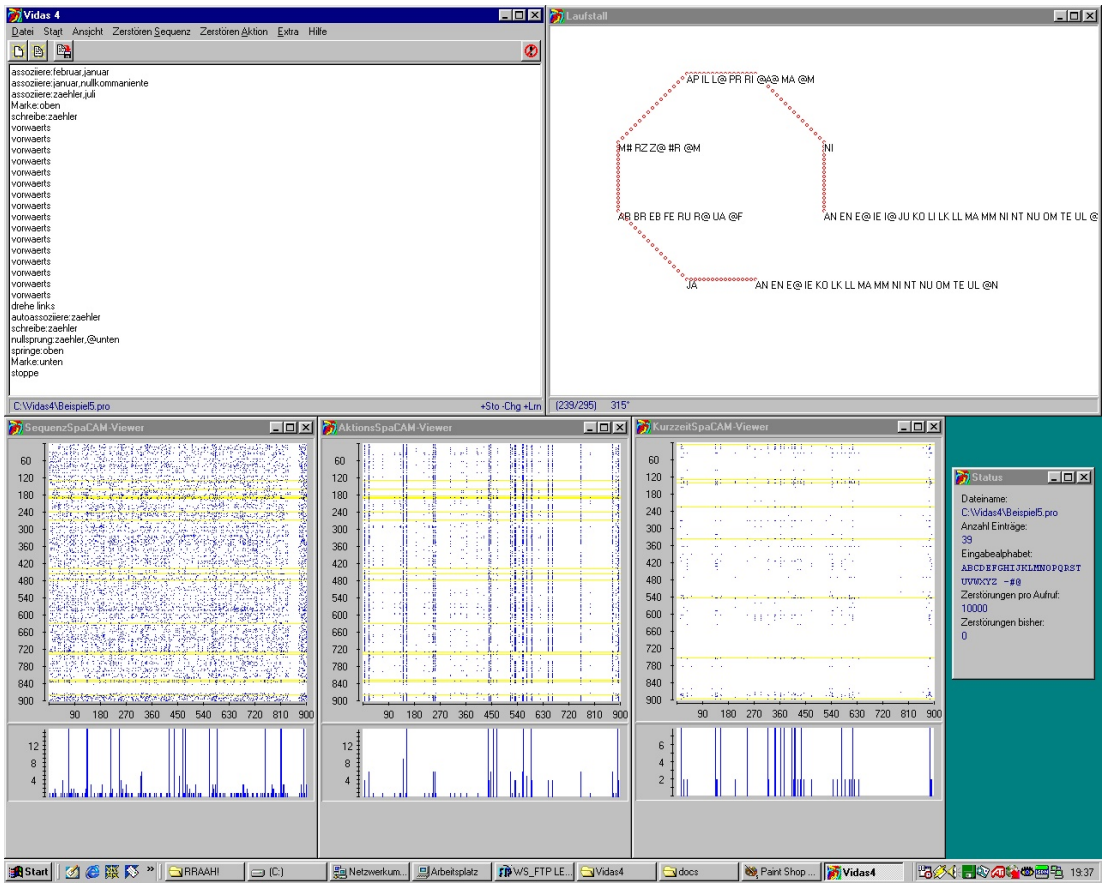


Abb. 46: Übersicht der Fenster von VIDAS 4

Nach Aufruf von VIDAS 4 erscheint der Programmeditor mit dem zuletzt bearbeiteten Programmtext. Nach Start dieses Programms (durch Druck auf die Taste F9) erscheinen auf dem Bildschirm bis zu sechs Fenster (s. Abb. 46): der Programmeditor (links oben), das Ausgabefenster (rechts oben), drei Fenster mit der Darstellung des Zustands dreier Assoziativmatrix von VIDAS 4, deren aktuellen Schwellwerten (links unten) und das Statusbord (rechts unten).

Über die unter dem Titelbalken liegende Steuerleiste des Editors (s. Abb. 46, rechts oben) erhält man durch Menüs (Menüzeile oben mit den Einträgen Datei, Start, Ansicht, Zerstören, Extra und Hilfe) und einer Reihe von Schaltflächen (angereiht auf einer Leiste darunter) Zugang zum Funktionsumfang von VIDAS 4. Den Schaltflächen (Buttons) auf der Leiste sind (von links nach rechts aufgezählt)

folgende Funktionen zugeordnet: Neue Programmdatei anlegen, Programmdatei öffnen, Programmdatei speichern und VIDAS 4 beenden.

Dem Statusbord kann man wie in VIDAS 3 [87] Informationen zum Zustand der aktuellen Abfolge-Assoziativmatrix A entnehmen. Neben dem Dateinamen der Programmdatei ist es die Anzahl der gelernten Assoziationspaare, das Alphabet, über dem kodiert wurde, wie viele Zerstörungen pro Aufruf vorgenommen werden und wie viele Zerstörungen bisher bereits passiert sind.

Im Ausgabefenster (s. Abb. 46, oben links) führt eine gedachte Figur („Turtle“), dargestellt durch einen kleinen roten Kreis, die ihm durch das Programm übermittelten Befehle aus. Dazu gehören Vor- und Rückwärtsbewegungen, Drehungen, Schreib- und Sprechanweisungen. Im gezeigten Beispiel sollte die Figur auf ihrem Weg ein Achteck ablaufen und an jedem Eckpunkt den Inhalt einer Variablen auslisten. Durch eine Doppelzeichenkodierung im Ausgang des Kurzzeitgedächtnisses erscheint der Variableninhalt in Doppelzeichen zerlegt (und müsste zum Lesen durch den Menschen rekombiniert werden).[88] Eine genauere Beschreibung liefert Beispiel 5 in Kap. 4.7.7 und Abb. 55.

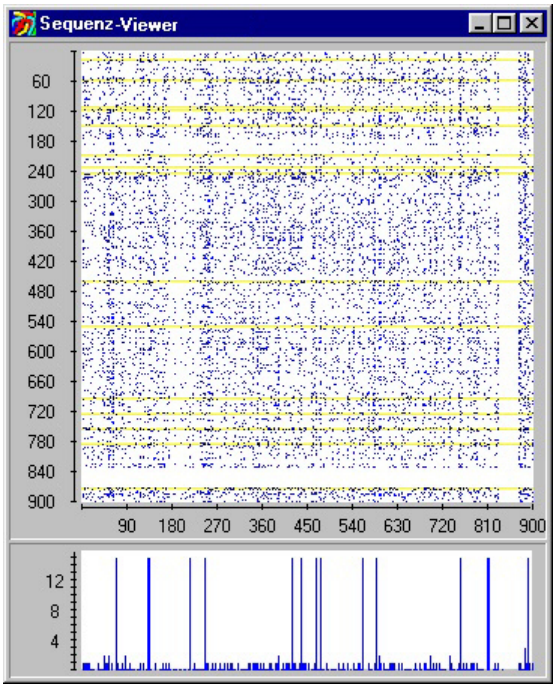


Abb. 47: Assoziativmatrix-Darstellung bei VIDAS 4

[87] s. Abb. 29
[88] Zur Doppelzeichenkodierung s. Kap. 8

Im oberen Teil des Assoziativmatrix-Viewers (s. Abb. 47) wird die Verteilung der Bits der Assoziativmatrix dargestellt. Die Darstellung erfolgt um den Faktor 3 gestaucht.[89] Nach einer Abfrage sind diejenigen Zeilen der Assoziativmatrix gelb markiert, die während der Abfrage aktiviert wurden (Aktivitätsmuster).

Im unteren Teil des Viewers wird das zur letzten Abfrage gehörende Schwellwerttupel wiedergegeben. Die Skala am linken Rand des Teilfensters dient zum Ablesen der Schwellwerte.

Als Befehle stehen in VIDAS 4 folgende zur Verfügung:

- vorwaerts
- rueckwaerts
- drehe {links | rechts}
- schreibe: <Bezeichner>
- sprich: <Bezeichner>
- assoziiere: <Bezeichner>, <Bezeichner>
- autoassoziiere: <Bezeichner>
- Marke: <Sprungziel>
- Springe: <Sprungziel>
- Nullsprung: <Bezeichner>, <Sprungziel>
- stoppe

Neben einer einfacheren Syntax wurde als wichtige Änderung zum Befehlsvorrat von VIDAS 3 der Befehl 'autoassoziiere' eingefügt, durch den die Kurzzeitmatrix K mit dem Wert von <Bezeichner> abgefragt und der resultierende Wert als neuer Wert von <Bezeichner> in K eingetragen wird. So wird es möglich, mit Hilfe von Assoziationsketten Schleifenstrukturen zu bilden.[90]

4.7.7 Beispiele

Beispiel 1: Sequenz aus Vorwärts- und Drehbewegungen

Die Turtle soll auf dem Ausgabebildschirm ein regelmäßiges Achteck zeichnen (zum Schluss als Markierung des Endes einen Haken nach links). Jede Seite besteht aus zehn Vorwärtsschritten. Dadurch entsteht ein Programmtext mit 93 Programmzeilen der Gestalt:

[89] Es werden jeweils drei Matrixreihen in einer Pixelreihe angezeigt.
[90] s. Kap. 4.7.7, Beispiel 4, und Kap. 5 zur Assoziativen Programmierung

```
vorwaerts
vorwaerts
vorwaerts
vorwaerts
vorwaerts
drehe rechts
vorwaerts
vorwaerts
...
```

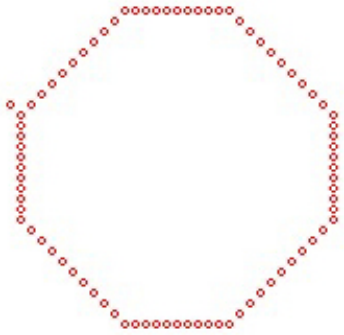


Abb. 48: Beispiel 1: 93 Programmschritte für ein Achteck

Die Ausführung des Programms ergibt das erwartete Achteck im Ausgabefenster (s. Abb. 48). Startpunkt der Figur ist stets die Mitte dieses Fensters. Ihre Richtung zeigt beim ersten Programmstart stets nach oben.

Der Zustand der Befehls-Matrix nach dem Lernen wird in Abb. 49 sichtbar. Da in diesem Beispiel viele gleichlautende Befehle (80 mal 'vorwaerts') gegeben wurden, verrät der Zustand der Befehls-Matrix durch die deutlich erkennbaren vertikalen Linien diese Gleichförmigkeit.

Auch nach einer Viertelmillion Zerstörungsaktionen in der Abfolge-Matrix A (hier durch das Setzen von je 10.000 Bits in Kreisform an einem zufällig gewählten Ort) wurde das Programm noch fehlerfrei abgearbeitet (s. Abb. 50). Danach kam es typischerweise zu Ablauffehlern (s. Abb. 51), durch die die Figur nicht mehr wie vorgesehen ihren Weg nahm (ein Drehbefehl wurde nicht ausgeführt) oder in einigen Fällen auch stehen blieb oder sich in einer Endlosschleife verfing.

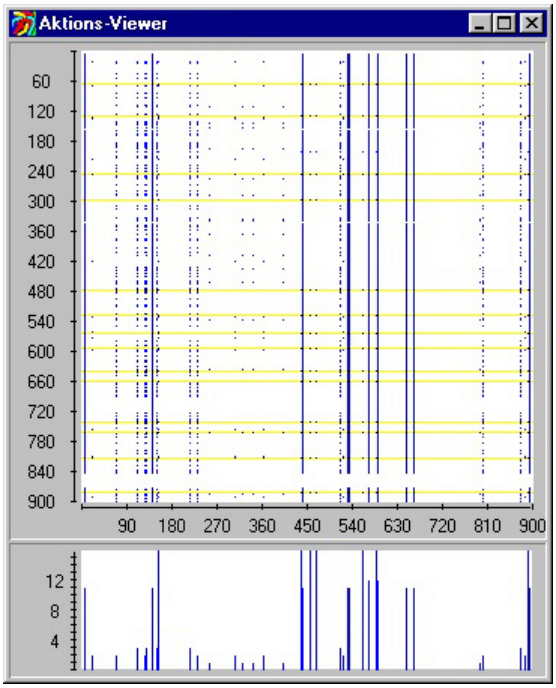


Abb. 49: Beispiel 1: Zustand der Befehls-Matrix nach dem Lernen

Beispiel 2: Absolute Sprünge

```
vorwaerts
vorwaerts
vorwaerts
Springe: weiter1
drehe rechts
vorwaerts
vorwaerts
Marke: weiter1
drehe rechts
vorwaerts
vorwaerts
vorwaerts
vorwaerts
```

Schon zur Lernzeit werden absolute Sprünge wie in obigem Programmtext, dadurch umgesetzt, dass der Zeilenschlüssel der Sprungbefehl-Programmzeile mit dem Zeilenschlüssel der Zeile assoziiert wird (in der Abfolge-Matrix A), die auf die Programmzeile mit der angesprungenen Marke folgt. Die übersprungenen Befehle werden nicht ausgeführt, gleichsam „auskommentiert“, es sei denn, sie enthalten

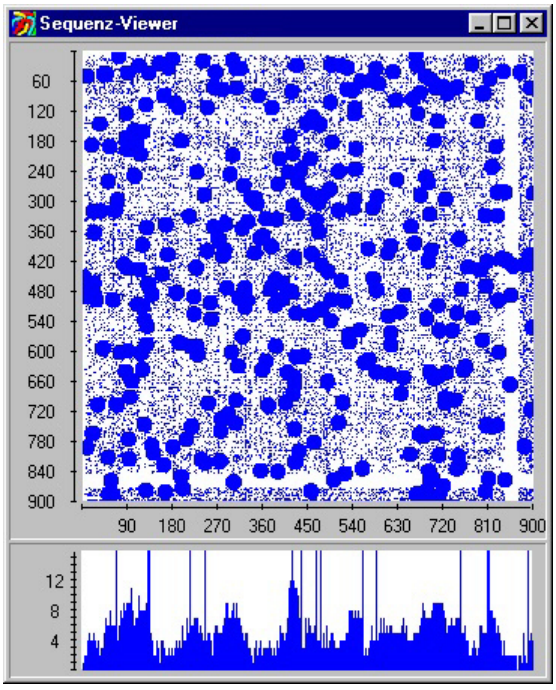


Abb. 50: Beispiel 1: Zerstörte Abfolge-Matrix

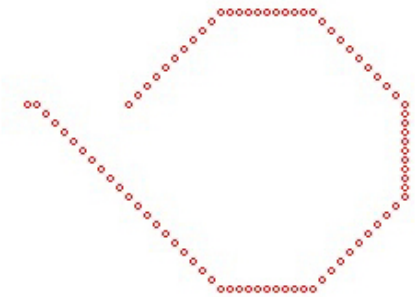


Abb. 51: Beispiel 1: Ein Drehbefehl wurde übergangen

eine Sprungmarke, zu der von anderer Stelle aus der Programmlauf verzweigt, was in diesem Beispiel aber nicht der Fall ist.

Beispiel 3: Variablen/Kurzzeitgedächtnis

```
Assoziiere: Rosenkohl, Gemüse
Vorwaerts
Vorwaerts
Vorwaerz
Vorwaerts
Drehe links
Drehe lings
Schreibe: Rosenkohl
Vorwaerts
Vorwaerts
Vorwaerts
Schreibe: Blumenkohl
Vorwaerts
Vorwaerts
Vorwaerts
Schreibe: Rhabarber
Stoppe
```

Die Schreibfehler in vorstehendem Programmtext sind beabsichtigt, um die Schreibweisentoleranz des Systems zu prüfen.

Zur Laufzeit eines VIDAS 4-Programms muss es möglich sein, Variablen mit Werten zu belegen und diese auch wieder abzufragen. Dafür steht ein Kurzzeitgedächtnis K in Form einer weiteren Matrix bereit. Würde man jedoch mit dem gleichen Variablenbezeichner stets neue Werte assoziieren (wie beispielsweise bei einer Schleifenvariablen), dann ergäbe die Abfrage mit diesem Variablenbezeichner eine Anhäufung aller bisher mit ihm assoziierten Werte auf gleicher Schwelle. Daher erfährt die Matrix, die das Kurzzeitgedächtnis repräsentiert, vor jedem Eintrag eines neuen Bezeicher/Wert-Paares eine Löschung aller Bits in den zum Bezeichner gehörenden Matrixzeilen.

Auch Zeilenschlüssel können in Variablen abgelegt werden, um zu gegebener Zeit zur so adressierten Zeile zu springen. Syntaktisch werden solcherart Variablenwerte durch ein vorangestellte '@'-Zeichen gekennzeichnet. Die hinter diesem Zeichen stehende Zeichenfolge muss irgendwo im Programmtext als Markenbezeichner auftauchen, damit die Programmzeile und ihr Schlüssel eindeutig bestimmt werden können.

Die in diesem Beispiel gelernte Assoziationspaar in der ersten Programmzeile führt dazu, dass Begriffe, die die Silbe 'kohl' enthalten, als Gemüse eingestuft werden. Der ebenfalls nicht gelernte Begriff 'Rhabarber' führt aber zu keiner Antwort auf hoher Schwelle (s. Abb. 52). Auch Leerzeichen sind beim Eintragen

von Bezeichner/Werte-Paaren relevant und werden ebenfalls in das Kurzzeitgedächtnis eingetragen.

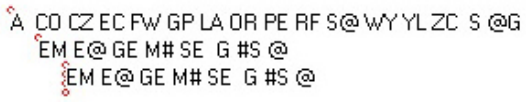


Abb. 52: Beispiel 3: Ergebnisse der Abfrage des Kurzzeitgedächtnisses

Der Zustand der Kurzzeitgedächtnis-Matrix nach Abfrage nach dem Begriff 'Rhabarber' zeigt, dass der Schwellwert bei 1 liegt (s. Abb. 53).

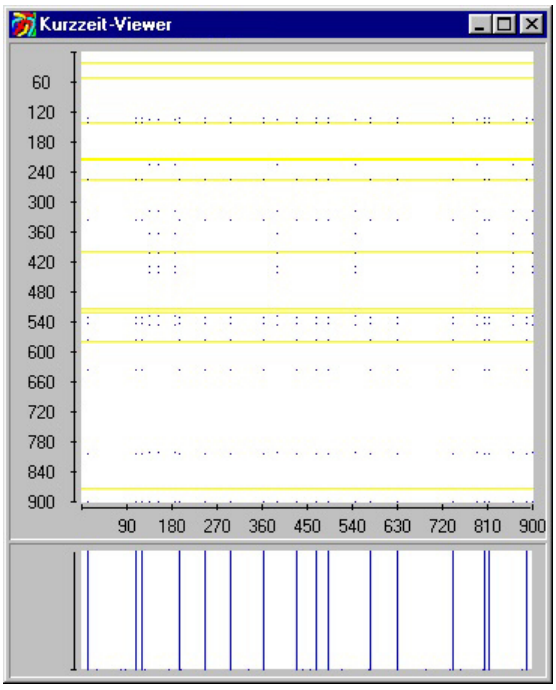


Abb. 53: Beispiel 3: Zustand der Kurzzeit-Matrix

Beispiel 4: Zähler/Bedingter Sprung

Das folgende Beispielprogramm sorgt für die Ablage einer Folge von Begriffspaaren im Kurzzeitgedächtnis K. Eine Schleifenvariable namens 'zaehler' erhält den Startwert und assoziiert sich bei jedem Schleifendurchlauf mit sich selbst

durch den Autoassoziiere-Befehl. Den Schleifenabbruch besorgt die Nullsprung-Anweisung, wenn 'zaehler' einen mit der Null zu identifizierenden Inhalt aufweist. In diesem Falle wird der Abfolgematrix A ein Zeilenschlüssel vorgelegt, bei dessen zugehöriger Programmzeile der Ablauf fortzusetzen ist.

```
Assoziiere:zehn,neun
Assoziiere:neun,acht
Assoziiere:acht,sieben
Assoziiere:sieben,sechs
Assoziiere:sechs,fünf
Assoziiere:fünf,vier
Assoziiere:vier,drei
Assoziiere:drei,zwei
Assoziiere:zwei,eins
Assoziiere:eins,nullkommaniente
Assoziiere:zaehler,zehn
Marke: oben
  Vorwaerts
  Vorwaerts
  Schreibe:zaehler
  Autoassoziiere:zaehler
  Nullsprung:zaehler, @unten
  Vorwaerts
Springe: oben
Marke: unten
Stoppe
```

Abb. 54: Beispiel 4: Abfolge der Werte der Zählvariablen

Das Kurzzeitgedächtnis K zeigte dann keine fehlerhaften Assoziationen, wenn der Ähnlichkeitsabstand der in ihm abgelegten Begriffe bezüglich der gewählten Kodierung (hier: Doppelzeichen) groß genug war (s. Abb. 54). Das nächste Beispiel zeigt das problematische Verhalten, wenn der Ähnlichkeitsabstand geringer ist.

Beispiel 5: Zählerwerte mit zu geringem Ähnlichkeitsabstand

```
assoziiere:dezember,november
assoziiere:november,oktober
assoziiere:oktober,september
assoziiere:september,august
assoziiere:august,juli
assoziiere:juli,juni
assoziiere:juni,mai
assoziiere:mai,april
assoziiere:april,märz
assoziiere:märz,februar
assoziiere:februar,januar
assoziiere:januar,nullkommaniente
assoziiere:zaehler,juli
drehe rechts
drehe rechts
vorwaerts
vorwaerts
...
```

Wählt man als Abfolgewerte für Schleifenvariable welche, die einen zu geringen Ähnlichkeitsabstand zueinander haben, so kann das wegen des hier gewählten Variablenkonzepts dazu führen, dass die Werte nicht mehr korrekt memoriert werden.

Der Wert 'Januar' wird letztlich nur noch in Teilen wiedergegeben (s. Abb. 55). Dennoch liefert die Assoziationskette die Abbruchbedingung. Das Programm stoppt.

Beispiel 6: Sprechbefehl

```
assoziiere:test,cinque
sprich:test
sprich:test
sprich:rhabarber
stoppe
```

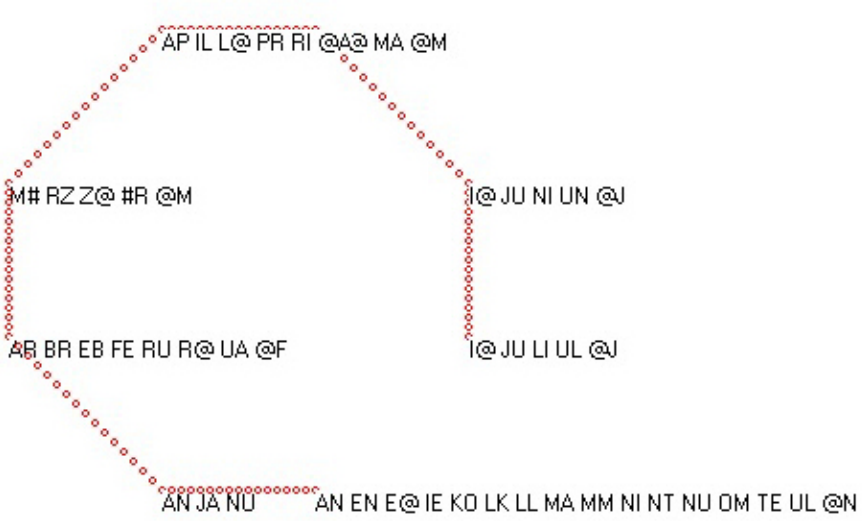


Abb. 55: Beispiel 5: Stopp der Schleife bei Nullwert

Liegen gewisse Variablenwerte vor, die eine Sprachausgabeeinheit im VIDAS 4-Modell mit ihr bekannten Wörtern verknüpfen kann, dann wird dieses Wort über die Soundkarte des Rechners ausgegeben, ansonsten ertönt ein kurzer Warnlaut. Zu diesem Zweck wird die Liste der Doppelzeichen, die das Kurzzeitgedächtnis K liefert, dahingehend überprüft, ob sie sich wieder zu dem vermeintlich ursprünglichen Wort rekombinieren lässt (mittels der Anfangs- und Endzeichen). Dieses Wort wird dann der Sprachausgabeeinheit vorgelegt.

4.7.8 Ergebnisse von VIDAS 4

In folgenden Punkten entsprach das VIDAS 4-Modell den Erwartungen:

i) Die Robustheit der Assoziativmatrizen zeigte sich auch bei längeren Abfolgen von Befehlen (Beispiel 1).

ii) Abfolgematrix A, Befehlsmatrix B und Kurzzeitgedächtnis K wirken beim Abarbeiten der Programmschritte reibungslos zusammen.

iii) Die schreibweisentolerante Eingabe des Programmtexts funktionierte, auch wenn eine Korrektur von Fehlschreibungen *vor* dem Eintragen in die Matrizen der Störfestigkeit dienlich wäre.

Die Experimente mit VIDAS 4 zeigten folgende Problemkreise auf, für die bei der zu entwickelnden VIDAS -Maschine Lösungen zu finden waren:[91]

[91] vgl. Kap. 4.9 und Kap. 5

a) Die zufallsgesteuerte Generierung von Programmzeilenschlüsseln führt gelegentlich dazu, dass ein Schlüssel einem anderen bezüglich seiner Kodierung für die Ablaufmatrix A zu sehr ähnelt. Dadurch sinkt die Störunanfälligkeit unnötig und es kommt unter Umständen zu Endlosschleifen.

b) Die beiden Befehle 'drehe rechts' und 'drehe links' unterscheiden sich durch ihre Bezeichnungen zu wenig (s. Abb. 49), so dass die Schwellwertmaxima zu nah beieinander liegen, um störunanfällig genug zu sein. Es ist eine andere Kodierung zu suchen.

c) Das Variablenkonzept des VIDAS 4-Modells ist unbefriedigend (Beispiel 5). Variablenbezeichner wie 'ZaehlerZwei' und 'ZaehlerDrei' sind sich zu ähnlich, so dass sich bei Wertzuweisungen und dem damit verbundenen Löschen des vorherigen Werts zu viele „Verluste“ bei den Werten der ähnlich bezeichneten Variablen ergeben.

d) Assoziationsketten werden mit dem besonderen Wert 'nullkommaniente' terminiert. Wenn nun aber genau dieser von einer Störung betroffen ist, findet die Assoziationskette ihr Ende nicht. Eine andere Darstellung der Null ist nötig.

e) Bedingte Sprünge im VIDAS 4-Modell finden ab und an ihr Ziel nicht, wenn sich bei Störungen der vom Kurzzeitgedächtnis K verantwortete Teil des Zeilenschlüssels nicht mehr gegen denjenigen der Abfolgematrix A ausreichend deutlich durchsetzt.

f) In einigen Abschnitten dieser Software-Simulation einer aufzubauenden VIDAS -Maschine entstand der Eindruck, dass zu sehr auf Möglichkeiten der Entwicklungsumgebung *Delphi* und des Wirtsrechners zurückgegriffen wurde, statt die Verwirklichung einer VIDAS -Maschine mit einfachen Mitteln im Auge zu behalten (z.B. Rekombinationsmöglichkeiten in Beispiel 6).

g) Um beim Eintragen der Programme spärlich zu kodieren, wurden Zeilenschlüssel, Variablen- und Befehlsbezeichner als Zeichenketten aufgefasst, die dann per Doppelzeichenkodierung in die Matrizen eingetragen wurden. Dieser umständliche Schritt ist hier im Grunde überflüssig. Er entstammt Programmiergewohnheiten aus vorangegangenen Projekten zum fehlertoleranten Textretrieval durch Assoziativmatrizen (auch aus dem Umgang mit Lerndateien in VIDAS 3). Eine direkte Kodierung durch die Lernumgebung einer VIDAS -Maschine erspart diesen Aufwand (s. Kap. 4.11).

Diese umfangreiche Liste an Mängeln von VIDAS 4 erlaubt die Einschätzung, dass diese Testumgebung erfolgreich war. Für die folgenden Schritte zur Verwirklichung der VIDAS -Maschine ergaben sich aus VIDAS 4 eine Vielzahl von Anregungen und Gelegenheiten zum Umdenken. Insbesondere der Mangel f) führte

zum Einsatz eines Digitalsimulators, der einfaches und exaktes Vorgehen einfordert.

4.8 Zusammensetzen der Matrix mit der Schwellwertlogik

Die Assoziativmatrizen der Hardwarelösungen aus Kap. 4.2 werden mit der Schwellwertlogik aus Kap. 4.4 versehen, um die Antworttupel zu den vorgelegten Fragetupeln zu ermitteln. Die Matrix und die Schwellwertlogik sind in Abb. 56 aus Gründen der Übersichtlichkeit in chipförmigen Bausteinen (Makros) links oben (Matrix) und in der Mitte unten (Schwellwertlogik) untergebracht. Die Schaltung in Abb. 56 stellt wiederum selbst ein Makro dar, dessen Ein- und Ausgänge durch quadratische Anschlüsse dargestellt sind.

Damit zwischen zwei aufeinanderfolgenden horizontalen Axonen, die eine „1“ liefern, der Zählerbaustein in der Schwellwertlogik dennoch seine Flanke (im Spannungs-Zeit-Diagramm) zum Weiterzählen bekommt, wird der Systemtakt invertiert an eine Zeile RS-Flip-Flops gelegt (s. in der Mitte von Abb. 56), die die Ausgangssignale der Matrix halten. Dadurch wird sichergestellt, dass zwischen je zwei Ausgabesignalen der Matrix die RS-Flip-Flops einmal zurückgesetzt werden. Zwei aufeinanderfolgende Einsen werden durch diese „Unterbrechung“ folglich beide von den Zählerbausteinen registriert.

Mit Hilfe des Digitalsimulators gelingt Entwurf und Veranschaulichung der aus Matrix und Schwellwertlogik zusammengefügten Schaltung. In Abb. 57 ist die Bedienoberfläche (Frontplatte, rechts) und die Schaltung (im Hintergrund) zu erkennen. Durch die Schalter oben und links auf der Frontplatte werden die Frage-/Antworttupel in die Matrix eingetragen. Für das hier wiedergegebene Beispiel waren das die drei Paare $(\mathfrak{f}_1, \mathfrak{a}_1)$, $(\mathfrak{f}_2, \mathfrak{a}_2)$, $(\mathfrak{f}_3, \mathfrak{a}_3)$ mit:

$$\mathfrak{f}_1 = (0,1,0,1,0,1,0,0,0,0,0,1,0,0,0,0)$$

$$\mathfrak{a}_1 = (0,1,0,0,1,1,0,0,0,0,0,0,1,1,0,0)$$

$$\mathfrak{f}_2 = (0,0,0,0,0,1,1,1,0,1,0,0,0,0,0,0)$$

$$\mathfrak{a}_2 = (1,0,0,1,0,1,1,0,0,0,0,0,0,0,0,0)$$

$$\mathfrak{f}_3 = (0,0,0,0,1,1,0,0,1,1,0,0,1,0,0,0)$$

$$\mathfrak{a}_3 = (0,0,0,0,0,1,1,1,0,1,1,0,0,0,0,0)$$

Nach dem Eintragen dieser drei Paare gemäß der Lernregel aus Kap. 2.2 sind die entsprechenden synaptischen Verbindungen der Matrix wie in Abb. 58 auf „1“ gesetzt (in der Grafik durch Verdickungen kenntlich gemacht).

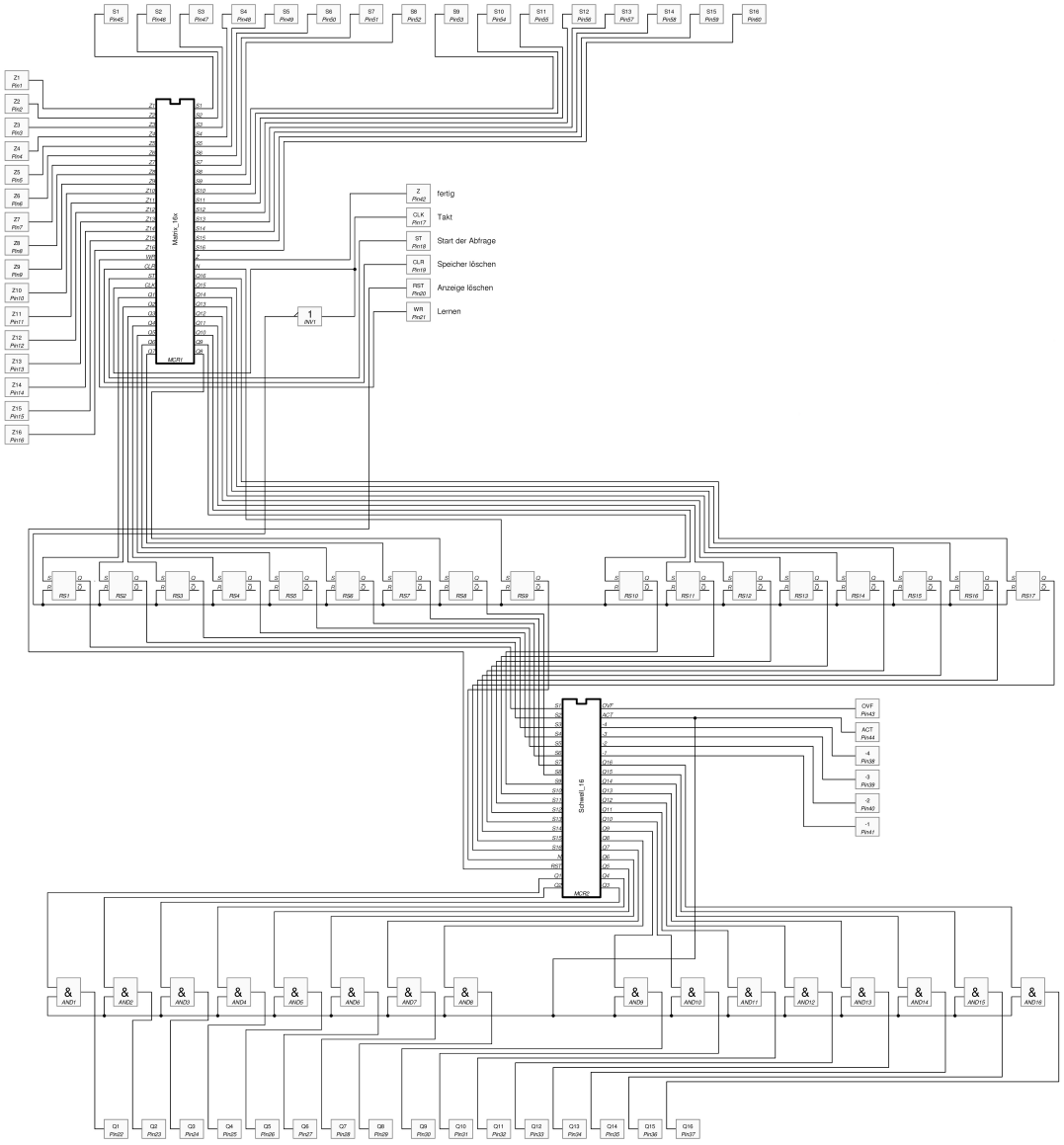


Abb. 56: Verbindung von Matrix und Schwellwertlogik

Fragt man die Matrix mit $\mathfrak{f}_1$, $\mathfrak{f}_2$, $\mathfrak{f}_3$ ab, ergeben sich die drei Schwellwerttupel:

$$\mathfrak{s}_1 = (1, 4, 0, 1, 4, 4, 1, 1, 0, 1, 1, 0, 4, 4, 0, 0)$$

$$\mathfrak{s}_2 = (4, 1, 0, 4, 1, 4, 4, 2, 0, 2, 2, 0, 1, 1, 0, 0)$$

$$\mathfrak{s}_3 = (2, 1, 0, 2, 1, 5, 5, 5, 0, 5, 5, 0, 1, 1, 0, 0)$$

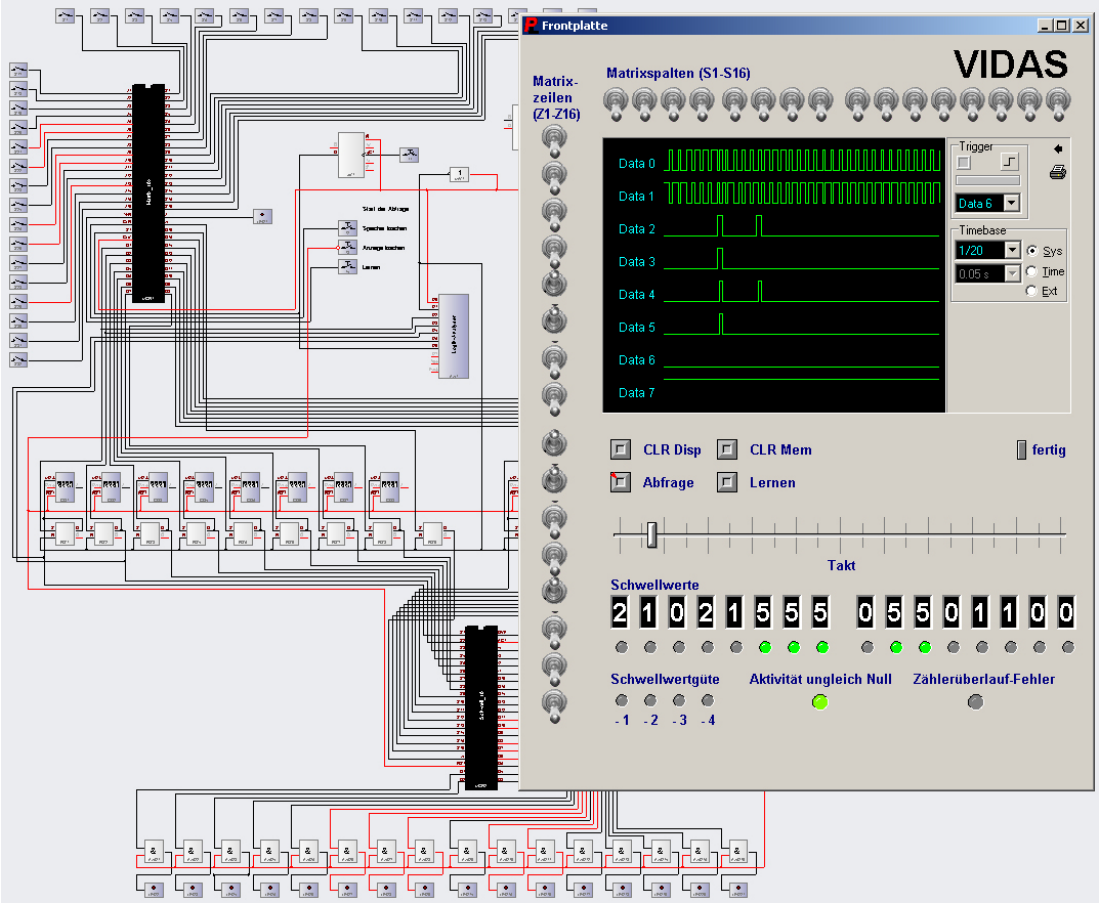


Abb. 57: Test der Hardware zu einer Assoziativmatrix

Abb. 59 demonstriert das Ergebnis für $\mathfrak{f}_2$. Die Schwellwerte werden durch Hexadezimalzähler auf der unteren Frontplattenhälfte angezeigt. Darunter leuchten Leuchtdioden bei den Schwellwertmaxima, die von der Schwellwertlogik bestimmt wurden. Die Leuchtdioden stehen für das Ergebnis der Abfrage, hier ist es (1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0), was genau dem Antworttupel $\mathfrak{a}_2$ entspricht. In der untersten Reihe der Frontplatte signalisieren Leuchtdioden die Schwellwertgüte (hier 100 %), ob im Fragetupel überhaupt eine Eins vorkam („Aktivität ungleich Null“) und ob es einen Zählerüberlauf gegeben hat. Das letzte Signal hat seinen Nutzen, wenn man an einer Stelle der VIDAS -Maschine prüfen möchte, ob die Kodierung noch spärlich genug ist.

Oberhalb der Anzeige der Schwellwerte lässt sich durch einen Schieberegler der Systemtakt einstellen, also die Geschwindigkeit, in der die Simulation durchgeführt werden soll. Die maximale Geschwindigkeit hängt von den Resourcen des

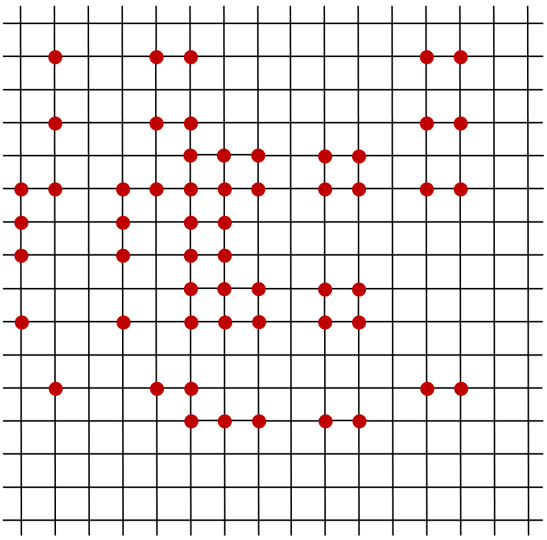


Abb. 58: Drei Muster gelernt

Wirtsrechners ab. Über dem Schieberegler dienen vier Taster zum Löschen der Anzeige der Schwellwerte („CLR Disp“), zum Löschen aller Flip-Flops der Matrix („CLR Mem“) und zum Abfragen und Lernen. Eine LED rechts meldet, ob eine Abfrage schon vollständig ausgeführt worden ist („fertig“).

In der oberen Hälfte der Frontplatte sind die Anzeigen des Logikanalysators von ProfiLab untergebracht. Dieser ermöglicht eine Einsicht in die zeitlichen Abläufe beim Lernen und Abfragen der Matrix. Auf dem Kanal „Data 0“ wurde der Systemtakt gelegt, darunter auf „Data 1“ der invertierte Takt, dessen Aufgabe in diesem Kapitel oben bereits beschrieben wurde. „Data 2“ und „Data 3“ zeigen die Spannungs-Zeit-Diagramme für die ersten beiden Matrixspalten (vertikale Axone). „Data 4“ und „Data 5“ geben ebenfalls die Signalverläufe an den ersten beiden Matrixspalten wieder, allerdings erst hinter der Zeile mit den RS-Flip-Flops. Man erkennt, dass eine Folge von drei aufeinanderfolgenden Einsen auf dem Kanal „Data 2“ im Kanal „Data 4“ mit drei voneinander getrennten Peaks auftaucht, um von den Digitalzählern als drei einzelne Einsen gezählt werden zu können.

Ersetzt man in dieser Versuchsanordnung das Makro mit der Assoziativmatrix durch ein Makro mit einer Matrix aus JK-Flip-Flops (vgl. Kap. 4.1) und lernt man die drei Paare $(\mathfrak{f}_1, \mathfrak{a}_1)$, $(\mathfrak{f}_2, \mathfrak{a}_2)$, $(\mathfrak{f}_3, \mathfrak{a}_3)$ in genau dieser Reihenfolge, dann ergeben sich die Schwellwerttupel

$$\mathfrak{s}_1 = (0, 3, 0, 0, 3, 4, 1, 1, 0, 1, 1, 0, 3, 3, 0, 0)$$

$$\mathfrak{s}_2 = (2, 0, 0, 2, 0, 4, 4, 2, 0, 2, 2, 0, 0, 0, 0, 0)$$

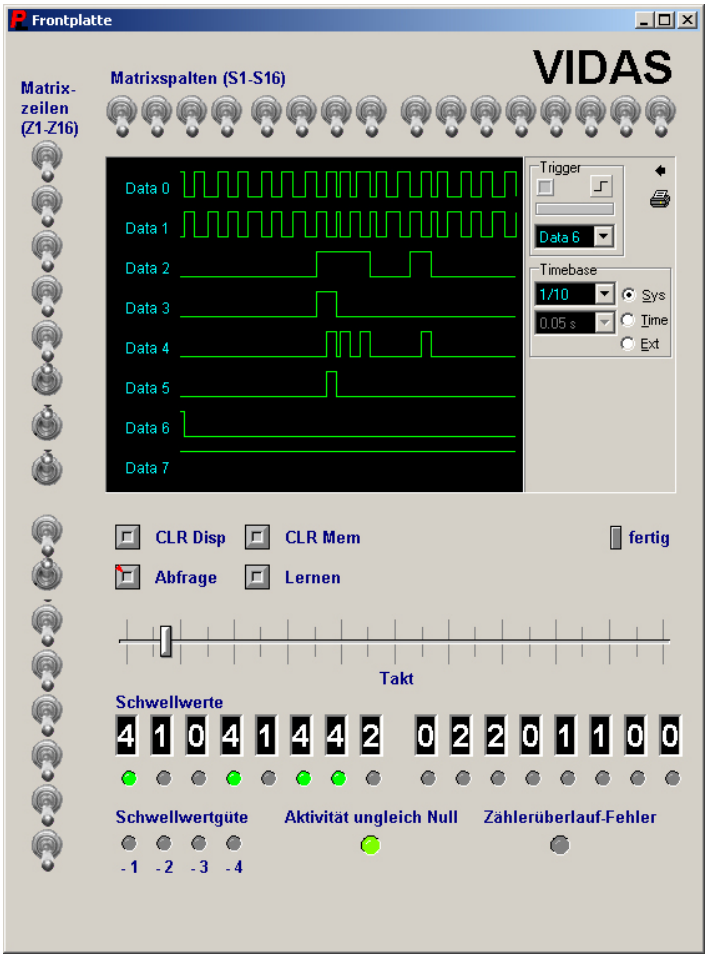


Abb. 59: Abfrage mit dem Muster $\mathfrak{f}_2$

$$\mathfrak{s}_3 = (0, 0, 0, 0, 0, 5, 5, 5, 0, 5, 5, 0, 0, 0, 0, 0)$$

und die Matrix wird wie in Abb. 60 belegt, wie sich nach Start der Simulation bestätigt. In der Belegung des Schwellwerttupels $\mathfrak{s}_3$ erkennt man die Wirkung der JK-Flip-Flops deutlich: außer den Schwellwertmaxima finden sich keine Einträge, alle anderen Werte wurden „vergessen“.

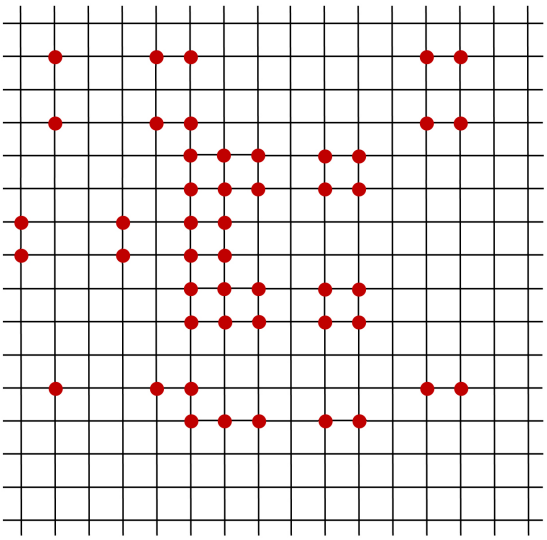


Abb. 60: Drei Muster in Matrix aus JK-Flip-Flops

4.9 Aufbau der VidAs -Maschinen VidAs 7 und VidAs 9

Die Entwicklung bis zu VidAs 7 mit fünf 8 x 8-Assoziativmatrizen diente zum Aufbau grundlegender Funktionen der Maschine. In dieser Größenordnung (gut 300 Flip-Flops) waren die Entwicklungszyklen noch kurz genug, um Konstruktion und Fehlersuche zügig abwickeln zu können.

Den Aufbau von VidAs 7 gibt Abb. 61 wieder. Ganz links befinden sich die Schnittstellen zum Dateisystem des Wirtsrechners, über die die Programme eingelesen werden. Ganz rechts liegen die Anzeigen des Systems (Leuchtdioden, Hex-Displays, Druckerschnittstelle). Im Planviertel oben links liegt die Abfolgematrix A (als Makro) mit all der Schaltlogik, die dafür sorgt, dass eine Programmzeile mit ihrer Nachfolgerin assoziiert wird. Dazu gehören auch Bausteine, die diesen Vorgang stoppen, wenn die Ausgangserregung der Matrix A auf Null sinkt. Die Maschine läuft dann zwar noch, führt aber kein Programm mehr aus. Taster rechts oberhalb von A erlauben die Eingabe einer beliebigen Programmzeile, ab der ein Programm ausgeführt werden soll. Im Unterschied zum VidAs 4-Modell (s. Kap. 4.7.4) wird der Prozess $p1$ (also das Assoziieren einer Programmzeile mit der nächsten) nur bei Bedarf unterbrochen. Die Falltür f aus Prozess $p3$ wird durch eine horizontale Reihe von RS-Flip-Flops verwirklicht (in Abb. 61 in der Mitte links).

Die Befehlsmatrix B aus dem VidAs 4-Modell wurde in drei Teile zergliedert, um ihre Aufgabe, außer Befehlen auch zugehörige Daten zu liefern, zu veranschaulichen. In einem etwaigen Platinenlayout würde man die Matrizen wieder

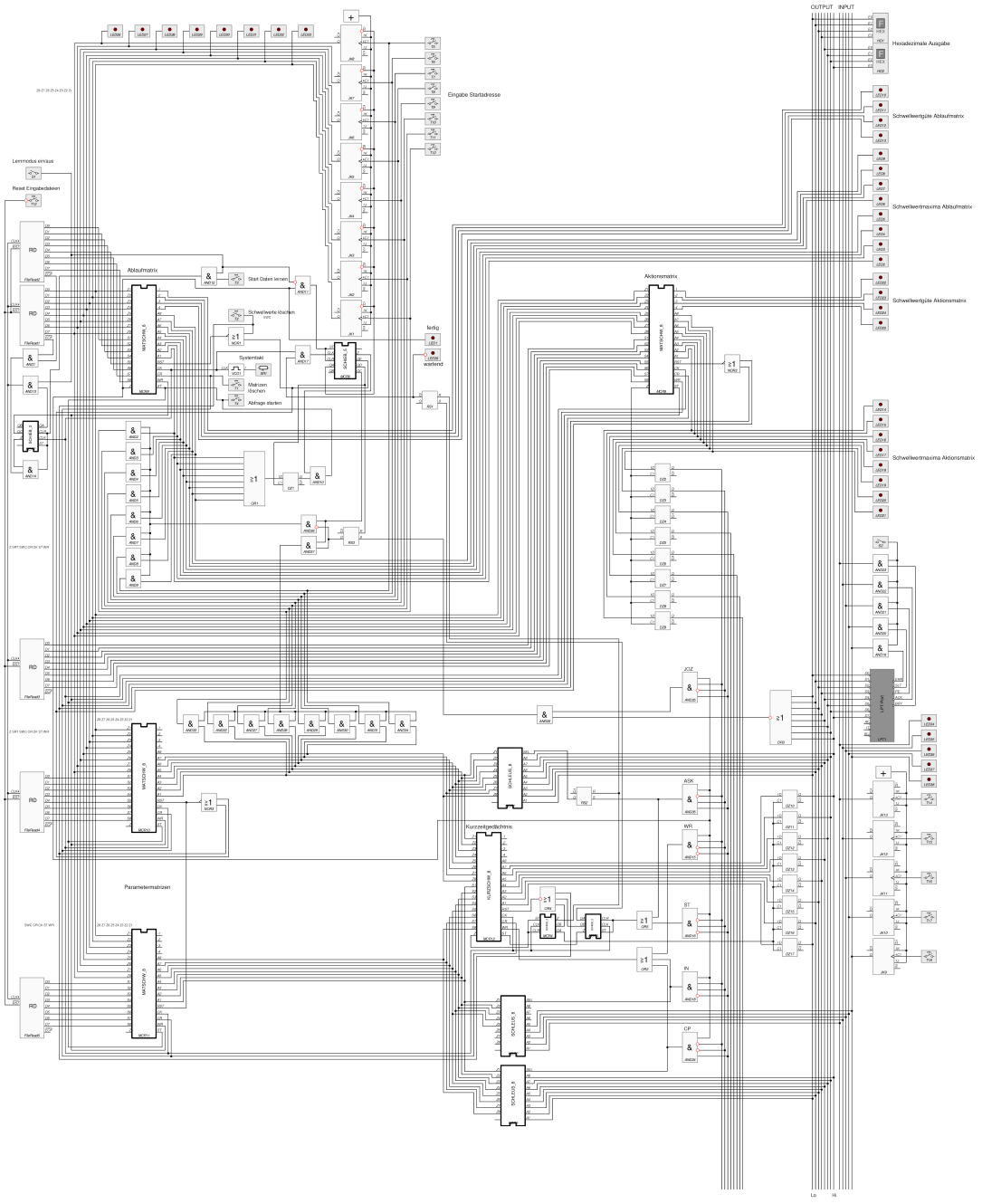


Abb. 61: VidAs 7

„zusammenstecken“.[92] Das Makro für die Befehlsmatrix B liegt im rechten oberen

[92] vgl. Kap. 4.2

Planviertel, die Makros für zwei Matrizen mit Daten zu den Befehlen (Parametermatrizen $P1$ und $P2$) im linken unteren Planviertel.

Das mit JK-Flip-Flops aufgebaute Kurzzeitgedächtnis K (im Planviertel von Abb. 61 unten rechts) wird von einer Reihe von Bausteinen umgeben, die auf Befehle von B warten, um die Lern- und Abfragevorgänge rund um K abzuwickeln. Da das Lernen mit einem Aufwand O(1) geschieht,[93] muss nur beim Abfragen von K die oben erwähnte Unterbrechungslogik für den Prozess $p1$ in Anspruch genommen werden.

Programme für die Maschine VIDAS 7 entstanden durch einen zugehörigen Programmeditor.[94] Mit diesem wurden die Dateien erzeugt, die über die Schnittstellen zum Dateisystem in den Matrizen $A, B, P1, P2$ die erforderlichen synaptischen Verbindungen einstellten. Im Umgang mit einer Reihe von Testprogrammen ergab sich, dass dem System ein Datenspeicher in Gestalt einer Matrix aus RS-Flip-Flops fehlt. Zwar könnte VIDAS 7 eine Reihe von Aufgaben wie etwa das Steuern einer einfachen Ampelanlage abwickeln, doch eine Möglickeit zur Aufnahme von Daten („Erfahrungen“) erweiterte das Anwendungsfeld deutlich. Diese Überlegungen fanden ihren Niederschlag in VIDAS 9 (s. Abb. 62).

VIDAS 9 besteht aus einem Verbund von sechs Assoziativmatrizen; die Ablaufmatrix A assoziiert mit einem Programmschritt den nächsten, Befehlsmatrix B liefert den zu einem Programmschritt gehörenden Befehl und die Parametermatrizen $P1$ und $P2$ die zugehörigen Parameterwerte. Ein- und Ausgaben werden über zwei Register abgewickelt, auf die Matrizen und etwaige Ein- und Ausgabe-Schnittstellenbausteine zugreifen können.

Das Langzeitgedächtnis L dient zur Laufzeit von VIDAS 9 als assoziativer Datenspeicher. Es ist eine Assoziativmatrix aus RS-Flip-Flops wie A, B, $P1$ und $P2$ auch. Das Kurzzeitgedächtnis K dient als Variablenspeicher und ist eine Assoziativmatrix, deren synaptische Verbindungen durch JK-Flip-Flops gebildet werden,[95] damit eine Variable ihren vorherigen Wert „vergessen“ kann.

Im Verschaltungsplan von VIDAS 9, so wie ihn ProfiLab zur Simulation anzeigt (s. Abb.63),[96] erkennt man oben als chipförmige Makros die vier Matrizen A, B, $P1$ und $P2$, darunter die horizontalen Axone dieser Matrizen. Da man diese vier Matrizen bei einem etwaigen Bau der Maschine zu einer einzigen Matrix zusammenstecken würde, verliefe dieses Leitungsbündel dann innerhalb der Matrizen und tauchte äußerlich nur noch bei einigen Anschlüssen auf. Die vertikalen Axone

[93] s. Kap. 4.3
[94] s. Kap. 4.11
[95] s. Kap. 4.1
[96] Eine größere Abbildung befindet sich im Anhang in Abb. 86.

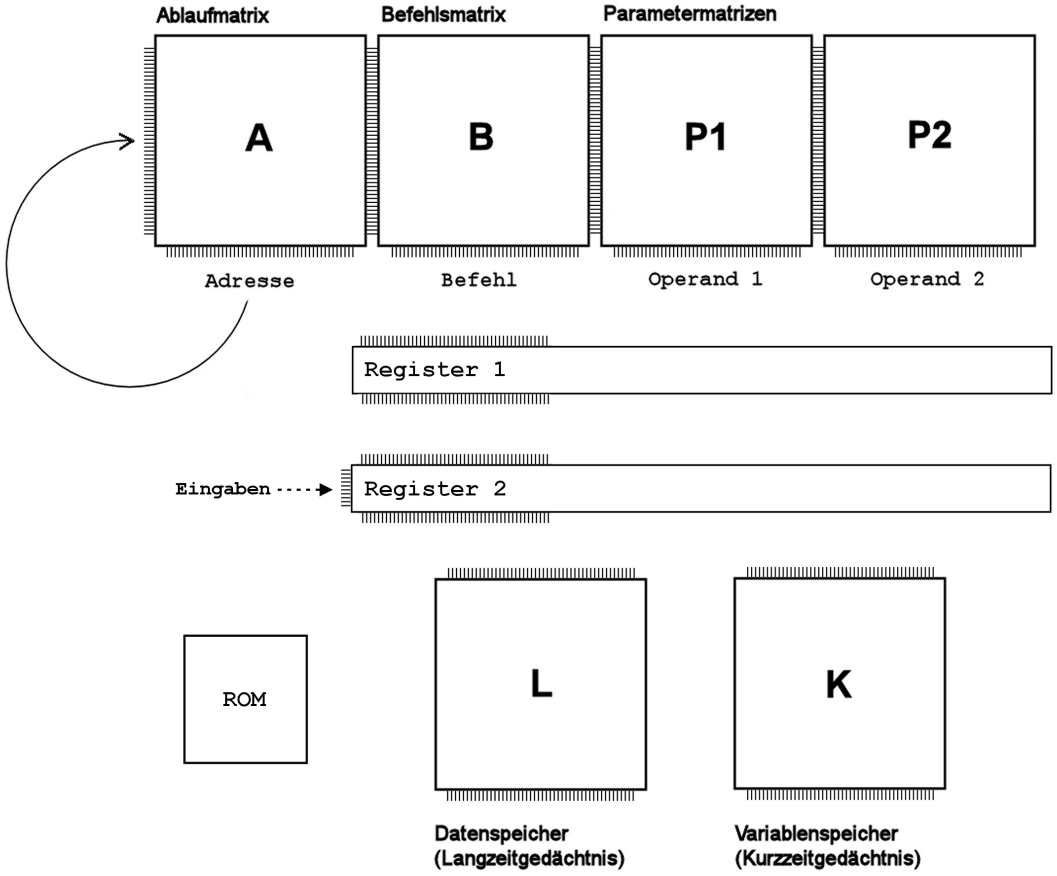


Abb. 62: Übersicht zu VidAs 9

sind aus Gründen der Anschauung getrennt nach ihren Aufgaben in vier vertikalen Leitungsbündeln zu sehen: links oben ein Bündel für die Abfolgematrix, in der Mitte links die Befehlsleitungen, an denen sich Dekodierbausteine die an sie gerichteten Befehle abgreifen, in der Mitte rechts die Datenleitungen von $P1$, von wo auch die horizontalen Axone von K und L bedient werden, und rechts die Datenleitungen von $P2$, die mit den vertikalen Axonen von K und L verbunden sind.

Unten rechts liegt die Matrix L und darüber der Variablenspeicher K. Zwischen beiden liegen Leitungsbündel, auf denen die Daten der beiden Register (vgl. Abb. 62) abgelegt werden. Auf die beiden Register greifen K und L in verschiedenerlei Weise zu (Ein-/Ausgaben, Vergleiche, Kopien). Links neben L wurde ein konventionelles ROM in die Schaltung eingebracht, so dass vorbereitete Muster, die in diesem ROM liegen, jederzeit programmgesteuert nach L herüberkopiert werden können. Für das Übertragen der ROM-Muster nach L sorgen Digitalzähler und

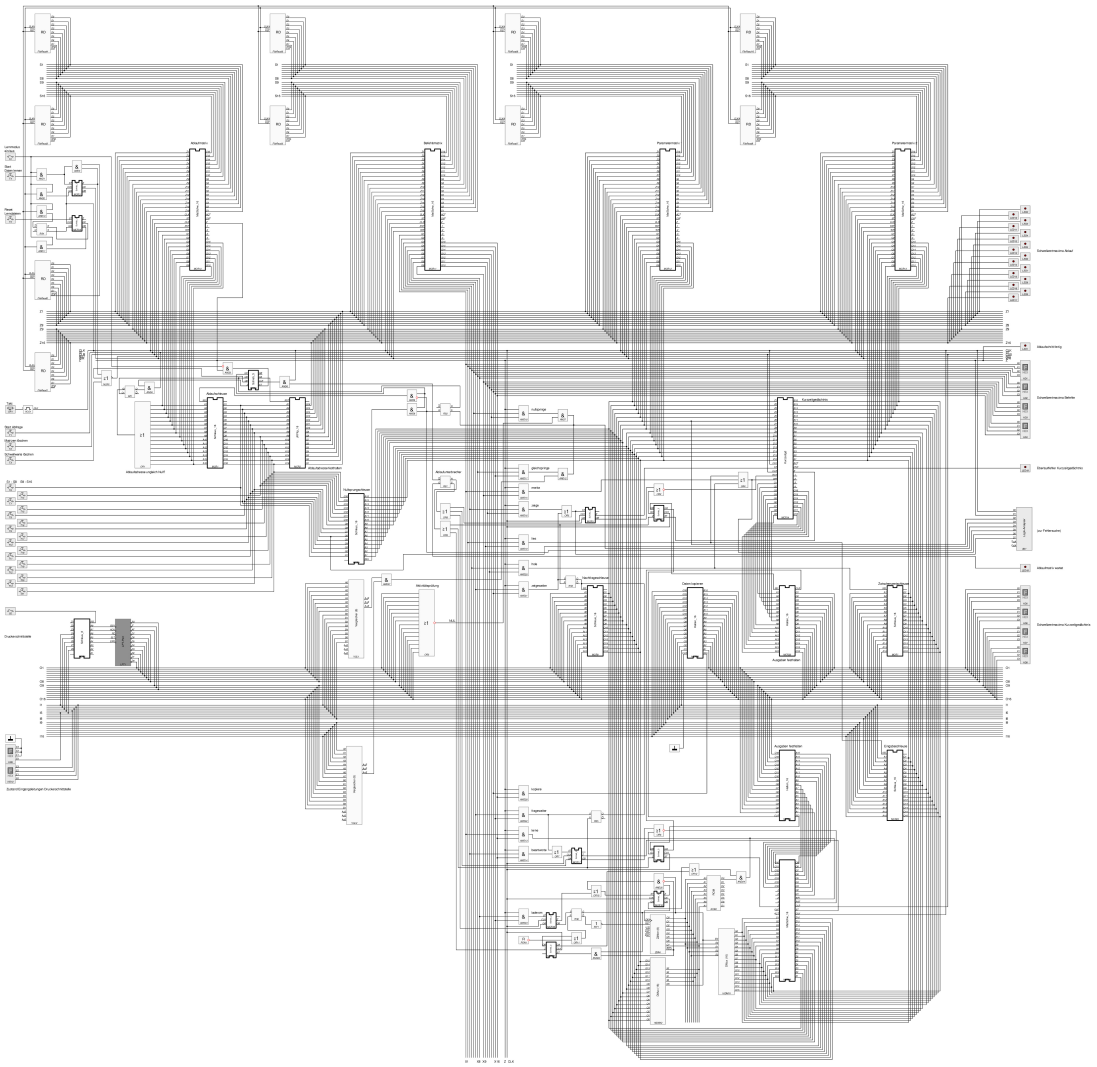


Abb. 63: VidAs 9

Demultiplexer. Für die Dauer der Übertragung wird, wie oben bereits für VidAs 7 beschrieben, der Prozess $p1$ unterbrochen.

Die Schnittstellenbausteine zum Dateisystem des Wirtsrechners, befinden sich am oberen Rand und oben ganz links im Plan. Um bei größeren Matrizen die Anzahl solcher Anschlussleitungen klein genug zu halten, wäre an den Schnittstellen mit entsprechenden Demultiplexern und mit seriellen Anschlussbausteinen zu arbeiten. Um extern den Befehlssatz zu erweitern, der fest in VidAs 9 eingebaut wurde (s. Kap. 5), könnte man die Befehls- und Datenleitungen ebenfalls mit Schnittstellenbausteinen nach außen führen.

4.10 Programme auf der VIDAS -Maschine

Abb. 64: Bedienfeld VIDAS 9

Nach dem Start der Simulation erfolgt im Digitalsimulator ProfiLab die Bedienung der Schaltung von VIDAS 9 [97] über eine Frontplatte (s. Abb. 64). Eine vertikale Reihe von Tastern links dient zum Einstellen der Bitfolge für die Abfolgematrix A, mit der der Programmlauf gestartet werden soll („Startadresse“). Eine dazu parallele Reihe von Leuchtdioden zeigt die jeweils aktive Bitfolge an. Es ist also möglich, einen Programmablauf auch beispielsweise in der Mitte zu beginnen, falls das einmal gewünscht wird. Der Programmeditor (s. Kap. 4.11) wählt bei der Kodierung des jeweiligen Programms eine Startadresse aus, die der Anwender hier eingibt.

Mit einem horizontalen Schieberegler in der Mitte reguliert man die Taktfrequenz. Unterhalb des Schiebereglers signalisieren zwei Leuchtdioden, ob die Ablaufmatrix A die nächste Programmzeile ermittelt hat ('fertig') oder ob der Programmablauf wegen eines zwischenzeitlichen Zugriffs auf das Kurz- oder Langzeitgedächtnis unterbrochen wurde ('wartend'). Eine weitere Leuchtdiode rechts meldet einen etwaigen Überlauf eines der Digitalzähler der Schwellwertlogik.[98]

Die hexadezimale Anzeigen der Codes des aktuell abgearbeiteten Befehls, des Zustands der Eingabeschnittstelle und der Ausgabeleitungen (Register 1) liegen in der Mitte der Frontplatte, darunter die Anzeige des Logikanalysators zur etwaigen Fehlersuche. Mit dem Schalter 'Druckerport ein' wird die Eingabeschnittstelle aktiviert.

[97] s. Abb. 63
[98] vgl. Kap. 4.4

Der Programmeditor erzeugt Dateien mit den Mustern für die Assoziativmatrizen $A, B, P1, P2$. In diesen Mustern steckt das jeweilige Programm. Das Programm, von dessen Ausführung Abb. 64 stammt, ist das folgende:

```
! Frage-/Antwortpaare eintragen

lerne $1212=$0101
lerne $3121=$1010
lerne $8080=$2222

! Abfragen

beantworte $1212
beantworte $3121
beantworte $8080

! Abfrage mit "gestörten" Fragen

beantworte $1213
beantworte $8082
beantworte $3180
beantworte $3112
```

Um die eben beschriebenen Muster eines Programms in die Matrizen zu übertragen, drückt man in der Mitte der Frontplatte zuerst auf die Tasten 'Matrizen löschen' und 'Schwellwerte löschen', um alle Flip-Flops des Systems zurückzusetzen, Reste der Muster eines vorherigen Programmlaufs also zu löschen. Dann schaltet man rechts 'Lernmodus ein' ein, um die Verbindung der VIDAS -Maschine mit dem Dateisystem herzustellen. Nach 'Reset Lerndateien' und 'Start Daten lernen' (Tasten in der Mitte oben) befinden sich die Muster in der Maschine, sobald keine der Leuchtdioden links mehr etwas anzeigt.

Nach dem Start mit obigem Programm werden von der Maschine nacheinander folgende Ausgaben angezeigt:

```
0101
1010
2222
0101
2222
1010
1111
```

Die Ausgaben entsprechen genau dem, was gemäß Lern- und Abfrageregel[99] für eine Assoziativmatrix (hier das Langzeitgedächtnis) zu erwarten ist. Mit den Befehlen 'lerne' und 'beantworte' werden Frage-/Antwortpaare in das Langzeitgedächtnis L eingetragen und abgefragt.

Die Liste der in der VIDAS -Maschine implementierten Befehle befindet sich in Kapitel 5. Dazu gehören Befehle zum Umgang mit Variablen im Kurzzeitgedächtnis K, für Frage-/Antwort-Paare im Langzeitgedächtnis L, für den bedingten Sprung und zum Umgang mit den beiden Registern. Alle Befehle wurden mit Hilfe von Testprogrammen wie dem obigen einzeln und im Zusammenwirken mit den anderen Befehlen auf ihre korrekte Funktion hin überprüft. Die Abarbeitung der Befehle auf der VIDAS -Maschine bespricht Kapitel 4.10.1. Wie die Maschine mit Sprüngen umgeht, erklärt Kapitel 4.10.2.

4.10.1 Abarbeitung von Befehlen zur Laufzeit

An den Befehlsleitungen, die von der Befehlsmatrix B ausgehen, überwachen Schaltgatter, ob ein sie betreffender Befehl in die Tat umzusetzen ist. Je nach gewünschter Fehlertoleranz wird über mehr oder weniger aufwändige Verknüpfungen von UND- und ODER-Gattern die Abarbeitung eines Befehls ausgelöst. In Abb. 65 dienen UND-Gatter der Entschlüsselung.

Abb. 65 ist ein Ausschnitt aus der Mitte des Schaltplans von VIDAS 9 in Abb. 63. Er soll als Beispiel für die Implementation von Befehlen dienen. Die zu den jeweiligen Befehlen gehörenden Schaltgatter sind entsprechend beschriftet. Oben erkennt man die Gatter für den Gleichspringe-Befehl. Über die Leitung CMP erhalten sie von einem Komparator die Auskunft, ob in Register 1 und 2 die gleiche Bitfolge liegt. Falls das so ist, wird die Sprunglogik aktiviert, die die aktuell in der Parametermatrix $P1$ befindliche Bitfolge der Abfolgematrix A als nächste Programmzeile übergibt.

Der darunter befindliche Baustein für dem Merke-Befehl setzt einen Schreibimpuls (WR) für das Kurzzeitgedächtnis frei, welcher die aktuellen Bitfolgen der Datenleitungen von $P1$ und $P2$ als Frage-Antwort-Paar in K einträgt.

Die Umsetzung des Zeige-Befehls ist aufwändiger als der Merke-Befehl, da zwar das Lernen in einem Taktschritt geschieht, das Abfragen wegen der Digitalzähler jedoch Zeile für Zeile nacheinander passieren muss. Daher wird bei Abfragen von K oder L zuerst die Unterbrecherlogik für den Prozess $p1$[100] eingeschaltet. Anschließend geht ein Abfrageimpuls (ST) an K. Sobald von K das Signal kommt,

[99] s. Kap. 2.2
[100] s. Abb. 41 und Abb. 43

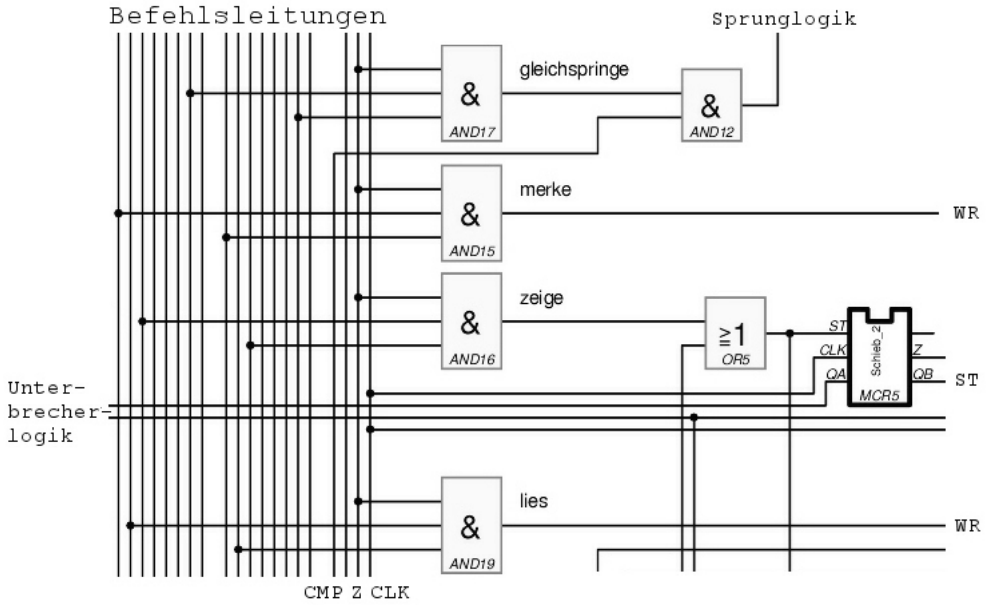


Abb. 65: Befehle in der VIDAS -Maschine

dass das Ergebnis vorliegt, wird die Unterbrecherlogik wieder ausgeschaltet. Diese Vorgänge regeln zwei Schieberegister, von denen eines (mit MCR5 beschriftet) hier abgebildet ist.

Das unterste Befehlsgatter sorgt dafür, dass Eingaben für den Lies-Befehl aus Register 2 im Kurzzeitgedächtnis K landen. Da das wieder einem Lernvorgang entspricht, genügt ein Schreibimpuls (WR) ohne Unterbrechung von $p1$.

Alle Befehlsgatter werden immer erst aktiv, wenn die Abfolgematrix über die Leitung Z meldet, dass der Zustand der Befehlsleitungen gültig ist. Damit wird verhindert, dass Zwischenergebnisse der Digitalzähler in der Schwellwertlogik schon Befehle auslösen. Die Versorgung mit dem Systemtakt erfolgt über die Leitung CLK.

4.10.2 Programmzeilenabfolge und Sprünge

Kernstück der Abarbeitung von Programmen durch die VIDAS -Maschine ist das Assoziieren einer Programmzeile mit ihrer Nachfolgerin. Abb. 66 gibt den betreffenden Ausschnitt des Schaltplans von VIDAS 9 wieder.[101]

Nachdem die Abfolgematrix A zur aktuellen Programmzeile die Folgezeile bestimmt hat, meldet sie dieses über die Leitung Z dem ganzen System und stößt

[101] s. Abb. 63, Mitte links

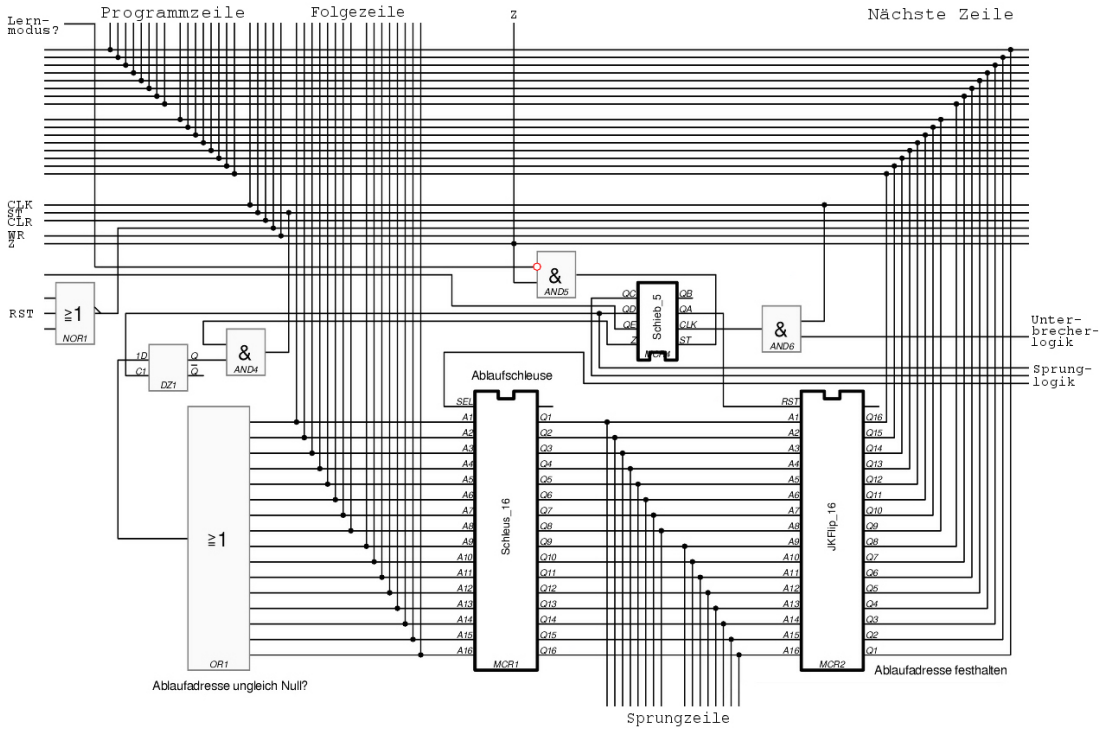


Abb. 66: Programmzeilenabfolge und Sprünge

gegebenenfalls über das UND-Gatter AND5 das Schieberegister Schieb_5 an, die weitere Abarbeitung der Situation zu übernehmen. Dieses ist an zwei Bedingungen geknüpft: erstens darf das System nicht im Lernmodus sein (AND5) und zweitens darf die Unterbrecherlogik die Versorgung mit dem Systemtakt nicht unterbrochen haben (AND6).

Zuerst setzt das Schieberegister Schieb_5 über QA alle JK-Flip-Flops im Makro MCR2 zurück, damit eine Bitfolge für die neue 'Nächste Zeile' nicht Reste einer alten Bitfolge enthält. Dann versucht Schieb_5 über QC die durch A gelieferte 'Folgezeile' an MCR2 durchzuschleusen (MCR1), was aber nur gelingt, wenn die Sprunglogik das zulässt. Liegt nämlich eine gültige 'Sprungzeile' an (s. Abb. 66, Mitte unten), dann wird diese als 'Nächste Zeile' in MCR2 eingetragen. Mit QD signalisiert Schieb_5, dass MCR2 eine 'Nächste Zeile' erhalten hat. Doch noch kann der Programmablauf nicht fortgesetzt werden. Zuerst prüft das ODER-Gatter, ob als 'Folgezeile' nicht eine Bitfolge von A kam, die nur Nullen enthielt, also das Programmende verlangt. In diesem Falle wird die Abfolgematrix A nicht neu gestartet (ST). Sonst werden alle Digitalzähler in der Schwellwertlogik auf Null gesetzt (QE) und die Abarbeitung des nächsten Programmschritts beginnt (Z).

4.11 Programmeditor für die VIDAS -Maschine

Für die Hardware von VIDAS 7 bis VIDAS 9 wurde jeweils ein Programmeditor geschrieben, der beim Eingeben des Programmtexts unmittelbar die Belegung der Assoziativmatrizen $A, B, P1, P2$ anzeigt (s. Abb. 67, oben rechts). Der Programmtext wird ins linke Fenster eingetragen und bei jeder Änderung wird unmittelbar sichtbar, wie sich dieses hinsichtlich der Assoziativmatrizen und Variablen auswirkt.

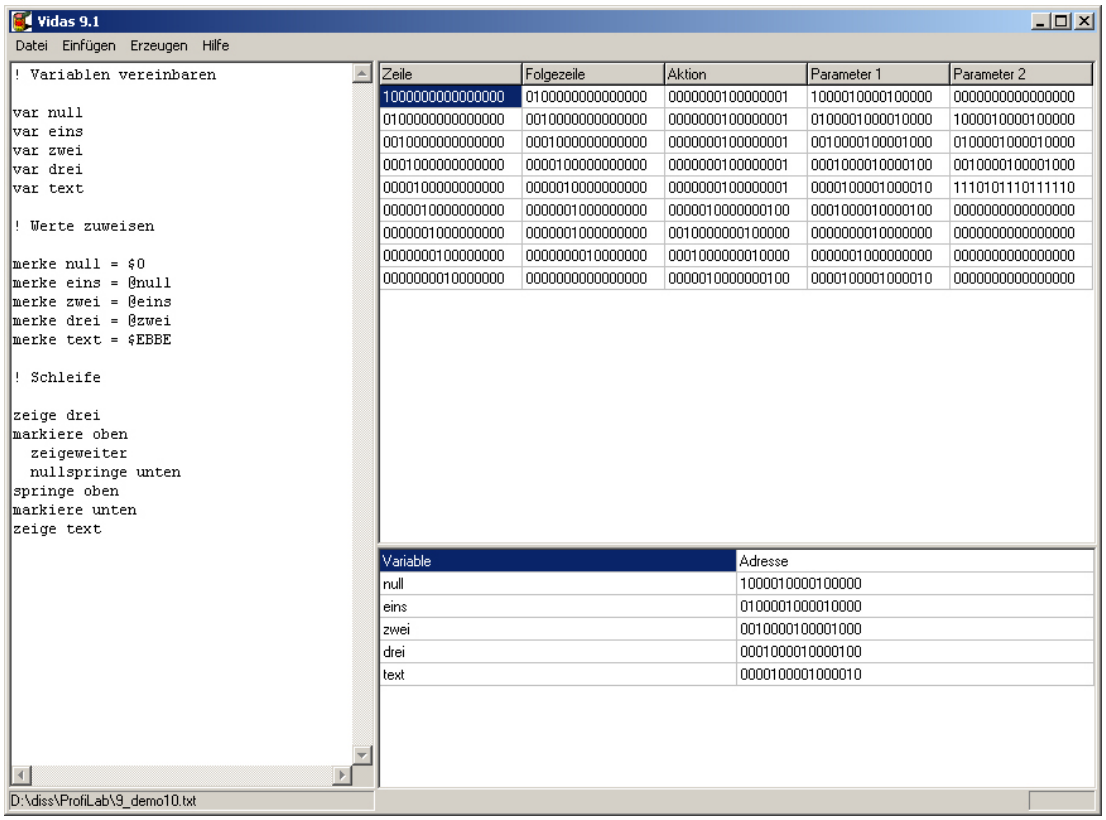


Abb. 67: Programmeditor von VIDAS 9

In der Übersicht über die in die Matrizen einzutragenden Werte (oben rechts) stehen in der ersten Spalte die Werte für die horizontalen Axone aller vier Matrizen. In den folgenden vier Spalten stehen dann die Werte für die vertikalen Axone der vier Matrizen $A, B, P1, P2$.

In einer Tabelle unten rechts erkennt man, welche Bitfolgen der Editor den vereinbarten Variablen zuordnet.

Der Editor legt durch Betätigen der Taste F9 die Belegungen für die Assoziativmatrizen in Dateien ab, aus denen sich die VIDAS -Hardware jederzeit während

der Simulation die Daten in die Matrizen $A, B, P1, P2$ laden kann.

Der ROM-Baustein, der der Matrix L zugeordnet ist, kann über ein eigenes Eingabefenster beliebig mit Werten gefüllt werden (Taste F8).

In einer Menüzeile unterhalb der Titelzeile des Programmfensters wählt man die gewünschten Tätigkeiten des Editors aus (Datei, Einfügen, Erzeugen, Hilfe). Ein Klick auf 'Einfügen' bietet eine Auswahl unter allen Befehlen, die zu VIDAS 9 gehören. Über den Menüpunkt 'Hilfe' kann man sich Erklärungen zu diesen Befehlen anzeigen lassen.

Die Hardware von VIDAS 9 arbeitet eine Reihe von Abfrage-, Lern-, Kopier- und Sprungbefehlen ab, so dass die Programme aus Sequenzen, Auswahlen und Wiederholungen aufgebaut werden können. Im Kapitel 5 werden der Befehlssatz von VIDAS 9 und diese Programme ausführlich beschrieben.

Beim Kodieren der Programmzeilen (Abb. 67, Spalten 'Zeile' und 'Folgezeile') und der Variablen (Abb. 67, Spalte 'Adresse') beachtet der Editor die im Kap. 2.2 aufgestellte Orthogonalitätsbedingung. Das bedeutet, dass je zwei Variablen in ihrer Darstellung als Bitfolge keine Eins gemeinsam am selben Platz besitzen, so dass das Skalarprodukt der Bitfolgen Null ergibt. Die Kodierung der Befehle (Abb. 67, Spalte 'Aktion') ist hardwareseitig vorgegeben; für sie gilt die Orthogonalitätsbedingung nicht.

5 Assoziative Programmierung

Was den herkömmlichen Computer seit den Anfängen seiner Entwicklung ausmacht, das ist die Möglichkeit, in vielfältiger Weise mit Zahlen umzugehen. Damit stehen heutige Computer in der Tradition von frühen mechanischen Rechenmaschinen wie etwa denen von Blaise PASCAL. Konrad ZUSE baute seinen Erfolg durch Maschinen auf,[102] die dank seiner Ideen zur Verarbeitung binär dargestellter Zahlen entstanden. Ein Rechenwerk gilt als Kernstück eines Rechners, wie dessen Name schon sagt. Programmzähler müssen beim Abarbeiten eines Programms voran- und zurückgerechnet werden, beim Speicherzugriff werden Adressen über Addierwerke bestimmt, Schleifen (Wiederholungen) werden gesteuert, indem das Rechenwerk Zähler auf die gewünschten Anzahlen bringt, und dergleichen mehr.

Die aus Assoziativmatrizen aufgebaute VIDAS -Maschine[103] hat kein Rechenwerk. Sie kann dennoch Programme abarbeiten. Sie kann dennoch mit Zahlen rechnen. Programmzähler braucht sie nicht, Speicheradressen sind nicht zu berechnen, Schleifen steuert sie über Assoziationsketten. Zudem ist sie wegen der Assoziativmatrizen störunanfällig und arbeitet fehlertolerant.

Der Programmierer einer solchen Maschine lässt sich darauf ein, dass Zahlen anders repräsentiert werden als durch eine binäre Zahlendarstellung, dass Adressberechnungen wie bei einem RAM-Speicher hier keinen Sinn machen, dass sich vieles in der Zuordnung von Fragen zu Antworten ausdrückt.

5.1 Register und Speicher

An Speichern stehen dem Programmierenden der Variablenspeicher K und der Datenspeicher L zur Verfügung. Zur Ablage von Zwischenwerten sind zwei Register vorgesehen (s. Abb. 68).

Der Datenspeicher L ist eine gewöhnliche Assoziativmatrix, der Variablenspeicher K wurde als Matrix konstruiert, die das Löschen vorheriger Variablenwerte erlaubt.[104] Eingaben an die Maschine erfolgen über Register 2. Im ROM werden Frage-/Antwortpaare (Muster) abgelegt, die zu einem beliebigen Zeitpunkt programmgesteuert nach L übertragen werden können.

[102] Die Z1 entstand 1938 als erster, frei programmierbarer Rechner der Welt um ein Gleitkommarechenwerk herum.
[103] vgl. Kap. 4.9
[104] vgl. Kap. 4.1

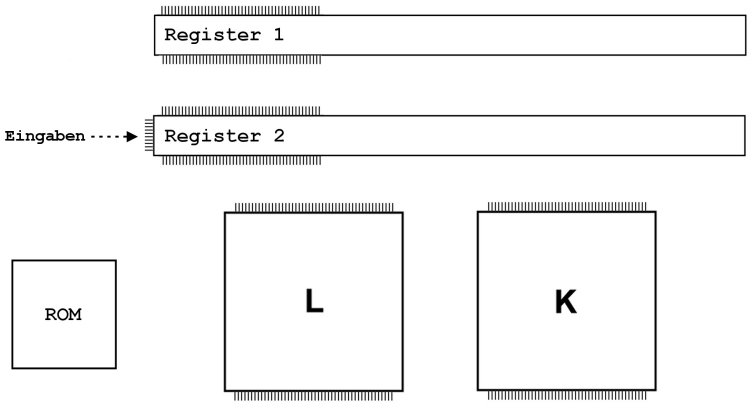


Abb. 68: Register und Speicher der VIDAS -Maschine

5.2 Variablenkonzept

Die Erfahrungen früherer Ansätze aufgreifend,[105] erhält jeder Variablenbezeichner nunmehr vom Programmeditor[106] eine geeignete Bitfolge zugeordnet. Der Variablenbezeichner ist nicht mehr selbst Zugriffsschlüssel auf den Wert der Variablen.

Variable	Darstellung
i	0000000000000000000000000000000000000000000000000000
j	000000040P00000*0000000000000000400000400000000
ja	0000*0000000020000800000000000000000000000000p0
nein	0001p0000000000000000000000000000*p0000000000

Abb. 69: Variablenverwaltung

Der Programmeditor zeigt die den Variablen zugeordneten Bitfolgen an (s. Abb. 69). Die Darstellung der Bitfolge geschieht zu einer frei wählbaren Basis (hier: 128-al-Darstellung). Durch Anklicken der Spaltenüberschrift (hier zum Beispiel auf 'Darstellung') wird die Spalte breiter angezeigt.

Die Vereinbarung von Variablen erfolgt über den Eintrag `var <bezeichner>` in irgendeiner Programmzeile. Als <bezeichner> kann eine beliebige Zeichenkette gewählt werden. Eine Typisierung der Variablen erfolgt nicht, da hier keine Speicherverwaltung wissen muss, wie viele Bytes für die Variable in einem RAM zu reservieren sind.

Beispiel: Die folgenden Programmzeilen bewirken die Tabelleneinträge von Abbildung 69.

[105] vgl. VIDAS 4, Kap. 4.7.8, Punkt c)
[106] vgl. Kap. 5.8

```
var i
var j
var ja
var nein
```

5.3 Befehlsvorrat von VidAs 9

Zwischen denjenigen Befehlen, die von der VidAs -Maschine abgearbeitet werden, und denjenigen, die sich an den Programmeditor richten, ist zu unterscheiden. In diesem Kapitel werden die Befehle vorgestellt, die in der Maschine[107] fest eingebaut sind. Im Kapitel 5.4 folgen dann die Befehle an den Editor.

5.3.1 Wertzuweisung an und Abfrage von Variablen

merke <bezeichner>=<wert>

Durch den Merke-Befehl wird der Variablen mit dem Bezeichner <bezeichner> der Wert <wert> ins Kurzzeitgedächtnis K eingetragen. Als <wert> gibt man eine Bitfolge an, die entweder binär, hexadezimal oder als ASCII-Zeichenfolge angegeben werden kann. Das erste Zeichen in <wert> legt die Darstellung fest: '*' bedeutet binär, '$' hexadezimal und ' " ' ASCII-Zeichenfolge. Ist das '@'-Zeichen das erste Zeichen in <wert>, wird der Variablen diejenige Bitfolge als Wert zugewiesen, die die Variablenverwaltung der anschließend in <wert> genannten Variablen zugeordnet hat. In anderen Programmiersprachen würde man eine so eingesetzte Variable als 'Pointer' bezeichnen.

Beispiel:

```
merke i = *010011001010
merke j = "Rosenkohl
merke k = $0AD89F012
merke l = @k
```

Die Variable i bekommt eine binär ('*') dargestellte Bitfolge zugewiesen, j erhält die Bitfolge als ASCII-Zeichenfolge, k wird der Wert hexadezimal übermittelt und die Variable l erhält diejenige Bitfolge als Wert, mit der die Variable k in der Variablenverwaltung dargestellt wird (vgl. Abb. 69).

zeige <bezeichner>

Den Wert einer Variablen mit dem Namen <bezeichner> legt der Zeige-Befehl ins Register 1. Dazu wird das Kurzzeitgedächtnis K (Variablenspeicher) mit der

[107] s. Abb. 63

Bitfolge abgefragt, mit der die Variablenverwaltung die Variable darstellt. Die tatsächliche Anzeige des Inhalts von Register 1 erfolgt bei der VIDAS -Maschine aus Kap. 4.9 über hexadezimale Displays; im Programmeditor aus Kap. 5.8 kann man die Anzeigedarstellung wählen.

Beispiel:

```
zeige i
zeige k
```

Ins Register 1 werden nacheinander die Werte von i und k eingetragen.

hole <bezeichner>

Für eine Zuweisung des Wertes in Register 1 an die Variable <bezeichner> sorgt der Hole-Befehl. Der Wert wird im Kurzzeitgedächtnis K abgelegt.

Beispiel:

```
hole i
hole k
```

Der Wert des Registers 1 wird nacheinander in die Variablen i und k eingetragen.

lies <bezeichner>

Für eine Zuweisung des Wertes in Register 2 an die Variable <bezeichner> sorgt der Lies-Befehl. Ins Register 2 werden Eingaben von etwaigen Peripheriegeräten eingetragen, daher rührt der Name dieses Befehls. Der Wert des Registers 2 wird im Kurzzeitgedächtnis K abgelegt.

Beispiel:

```
lies i
lies k
```

Der Wert des Registers 2 wird nacheinander in die Variablen i und k eingetragen.

5.3.2 Eintragen und Abfragen von Frage-/Antwortpaaren

lerne <frage>=<antwort>

Durch den Lerne-Befehl wird das Frage-Antwortpaar (<frage>,<antwort>) ins Langzeitgedächtnis L eingetragen. Für das Format von <frage> und <antwort> gelten die oben für den Merke-Befehl gegebenen Anmerkungen.

Beispiel:

```
lerne $12120120 = *01010101010101
lerne $0AD89F012 = "Blumenkohl
lerne "Eins = "Zwei
```

Der Frage $12120120 (hexadezimal dargestellt) wird die binär notierte Antwort *01010101010101 zugeordnet, der Frage $0AD89F012 die als ASCII-Zeichenfolge gegebene Antwort 'Blumenkohl' und der Frage 'Eins' die Antwort 'Zwei'.

beantworte <frage>

Der Beantworte-Befehl fragt mit <frage> das Langzeitgedächtnis L ab und legt die Antwort ins Register 1.

Beispiel:

```
beantworte $12120120
beantworte "Eins
```

Es werden nacheinander die Antworten auf die Fragen $12120120 und 'Eins' ins Register 1 gebracht.

5.3.3 Autoassoziationen

zeigeweiter

Mit dem Wert in Register 1 wird durch den Zeigeweiter-Befehl das Kurzzeitgedächtnis K abgefragt und die Antwort wiederum ins Register 1 gebracht. Das schafft die Möglichkeit, Pointer-Variablen auszuwerten.[108]

Beispiel:

```
var i
var j
var k
merke i = "Ende
merke j = @i
merke k = @j
zeige k
zeigeweiter
zeigeweiter
```

[108] Pointer — s. Merke-Befehl

In der Variablen k wurde die zu j gehörige Bitfolge eingetragen, in der Variablen j diejenige von i. Wenn man jetzt den Wert von k ins Register 1 bringt ('zeige k') und dann zweimal den Zeigeweiter-Befehl aufruft, liegt zum Schluss der Wert 'Ende' im Register 1.

frageweiter

Mit dem Wert in Register 1 wird durch den Frageweiter-Befehl das Langzeitgedächtnis L abgefragt und die Antwort wiederum ins Register 1 gebracht. Das schafft die Möglichkeit, Assoziationsketten zu folgen.

Beispiel:

```
lerne "Januar = "Februar
lerne "Februar = "Maerz
lerne "Maerz = "April
beantworte "Januar
frageweiter
frageweiter
```

Die Frage-/Anwortpaare ('Januar'/'Februar', 'Februar'/'Maerz', 'Maerz/April') werden in L eingetragen. Nach dem zweiten Frageweiter-Befehl liegt der Wert 'April' im Register 1.

5.3.4 Bedingte Sprünge

nullspringe <marke>

Um während des Programmablaufs zur Sprungmarke <marke> springen zu können, falls der Wert des Registers 1 „0“ ist, steht der Nullspringe-Befehle zur Verfügung. Sprungmarken werden vom Programmeditor verwaltet. Die Programmzeile, die auf <marke> folgt, wird bei einer „0“ in Register 1 als nächste ausgeführt. Steht die Sprungmarke <marke> am Programmende, stoppt das Programm.

Beispiel:

```
lerne $0422 = $3814
lerne $3814 = $8249
lerne $8249 = $0
beantworte $3814
nullspringe unten
frageweiter
nullspringe unten
frageweiter
markiere unten
```

Da auf die Abfrage von L mit \$3814 noch keine Null ins Register 1 gelangt, wird der erste Frageweiter-Befehl ausgeführt. Der liefert dann diese Null und das Programm stoppt. Damit die Abfrage von \$8249 tatsächlich eine Null liefert, muss sichergestellt sein,[109] dass in L in den Matrixzeilen, die durch \$8249 angesprochen werden, keine Einsen stehen.

gleichspringe <marke>

Für einen Sprung im Programmablauf zur Sprungmarke <marke>, falls die Werte von Register 1 und Register 2 übereinstimmen, dient der Gleichspringe-Befehl.

Beispiel:

```
lerne $0422 = $3814
lerne $3814 = $8249
lerne $8249 = $3814
beantworte $0422
kopiere
beantworte $8249
gleichspringe unten
beantworte $3814
markiere unten
```

Die Antwort von L auf die Frage \$0422 wird in das Register 1 gelegt und von dort mit dem Kopiere-Befehl (s.u.) ins Register 2 kopiert. Danach wird L mit \$8249 abgefragt und durch den Gleichspringe-Befehl festgestellt, dass die beiden Registerinhalte übereinstimmen (beide beinhalten den Wert \$3814). Der Programmablauf stoppt sogleich, mit der Frage \$3814 wird L nicht mehr abgefragt.

5.3.5 Registerkopie

kopiere

Durch den Kopiere-Befehl wird der Inhalt von Register 1 ins Register 2 kopiert. Beide Register beinhalten anschließend denselben Wert.

Beispiel:

```
lerne $3814 = $8249
beantworte $3814
kopiere
```

Nach Abarbeitung dieser drei Befehle befindet sich in Register 1 und 2 der Wert \$8249.

[109] Der Programmeditor in Kap. 5.8 leistet dieses auf Wunsch.

5.3.6 ROM-Kopie

laderom

Der Laderom-Befehl trägt den Inhalt des ROM in das Langzeitgedächtnis L ein. Das Langzeitgedächtnis wird vorher nicht gelöscht, sondern die im ROM gespeicherten Muster werden zum Bestand von L hinzugelernt.

Beispiel:

```
lerne $0422 = $3814
lerne $3814 = $8249
laderom
beantworte $0422
beantworte $3814
```

Zu den beiden in L abgelegten Frage-/Antwortpaaren ($0422, $3814) und ($3814, $8249) werden die Muster aus dem ROM hinzugeladen. Anschließend wird geprüft, ob die Antworten zu den beiden Fragen $0422 und $3814 noch die oben gelernten Antworten sind.

5.3.7 Pausen

pausiere

Um einen Programmablauf zu verlangsamen oder der Forderung nachzukommen, dass vor einem Springe-Befehl oder nach einem Markiere-Befehl ein ausführbarer Befehl aus diesem Kapitel stehen muss,[110] wählt man den Pausiere-Befehl.

Beispiel:

```
springe unten1
markiere oben1
pausiere
springe unten2
```

Ohne den Pausiere-Befehl würde hier der Markiere-Befehl keinen ausführbaren Befehl adressieren.

[110] s. Kap. 5.4.1

5.4 Befehlsvorrat des Programmeditors

5.4.1 Sprünge

markiere <marke>

Der Programmeditor verwaltet die Sprungmarken, die durch den Markiere-Befehl festgelegt werden. Als <marke> kann man eine beliebige (eindeutige) Zeichenkette benutzen. Es wird durch den Befehl die auf die Markiere-Programmzeile folgende Programmzeile markiert. Diese muss einen ausführbaren Befehl aus Kap. 5.3 enthalten.

Beispiel:

```
var i
merke i = "Blumenkohl
markiere oben
  zeige i
  nullspringe unten
springe oben
markiere unten
```

Da der Variablen i keine Null als Wert zugeordnet wurde, wird dieses Programm endlos 'Blumenkohl' anzeigen.

springe <marke>

Als absoluter Sprung bewirkt der Springe-Befehl eine unbedingte Verzweigung im Programmablauf zur Sprungmarke <marke>.

Beispiel:

```
var i
merke i = "Blumenkohl
springe unten
zeige i
markiere unten
```

Da der Sprung zur Marke 'unten' unbedingt erfolgt, wird der 'Blumenkohl' nie angezeigt. Absolute Sprünge können im Unterschied zu bedingten Sprüngen[111] schon zur Lernzeit des Programms fest eingeplant werden, ihre Kodierung für die Abfolgematrix A ist also eine Aufgabe des Programmeditors. Vor dem Springe-Befehl muss ein ausführbarer Befehl aus Kap. 5.3 stehen.

[111] s. Kap. 5.3.4

5.4.2 Assoziationsketten und -kreise

&afolge <varname>,<laenge>

Mit der Anweisung '&afolge' an den Programmeditor erzeugt dieser eine Assoziationskette im Langzeitgedächtnis L mit <laenge> Gliedern und legt die Startadresse in der Variablen <varname> ab. Dabei wird darauf geachtet, dass kein Element der Kette mit einem anderen Einsen gemeinsam hat, das heißt, je zwei verschiedene Fragetupel der Kette sind orthogonal zueinander. Das stellt sicher, dass der Assoziationskette störunfällig gefolgt werden kann. Das Ende der Assoziationskette wird durch eine Null markiert.

Beispiel:

```
var i
&afolge i,10
zeige i
markiere oben
  frageweiter
  nullspringe unten
springe oben
markiere unten
```

Eine Assoziationskette wird vereinbart und ihr „Kopf“ in der Variablen i gespeichert. Der Wert von i landet durch den Zeige-Befehl im Register 1. Anschließend leistet der Frageweiter-Befehl, dass nacheinander alle zehn Glieder der Kette abgelaufen werden, bevor das durch „0“ terminierte Ende der Kette erreicht ist und der Nullspringe-Befehl zur Marke 'unten' verzweigt.

&zfolge <laenge>

Durch diesen Befehl wird im Kurzzeitgedächtnis K eine Zahlen-Assoziationskette mit <laenge> Gliedern eingetragen. Die Folge der Variablenbezeichner beginnt mit 'eins', 'zwei', 'drei', usw. Das Ende der Kette kennzeichnet eine Null.

Beispiel:

```
&zfolge 10
zeige eins
markiere oben
  zeigeweiter
  nullspringe unten
springe oben
markiere unten
```

Neun Glieder der Kette werden angezeigt beginnend mit 'zwei', endend bei 'zehn'. Dann erreicht die Kette durch die abschließende Null ihr Ende.

&zkreis

Um eine Zahlen-Assoziationskette null – eins – zwei – ... – neun – null im Kurzzeitgedächtnis K zu erhalten, gibt man den &zkreis-Befehl ein. Diese Kette könnte man auch Ziffernkreis nennen.

Beispiel:

```
&zkreis
zeige neun
kopiere
zeige fünf
markiere oben
  zeigeweiter
  gleichspringe unten
springe oben
markiere unten
```

Die fünf Glieder des Ziffernkreises zwischen 'sechs' und 'null' werden angezeigt. Die Programmzeile 'zeige neun' bewirkt beim Ziffernkreis, dass die Darstellung der Null ins Register 1 kommt. Von dort wird sie mit dem Kopiere-Befehl ins Register 2 gebracht, um später mit dem Gleichspringe-Befehl auf diese Null reagieren zu können.

&rkreis

Um eine Richtungs-Assoziationskette nord – nordwest – west – ... – nordost – nord im Kurzzeitgedächtnis K zu erhalten, gibt man den &rkreis-Befehl ein. Diese Kette könnte man auch Richtungskreis nennen.

Beispiel:

```
&rkreis
zeige nordost
kopiere
zeige süd
markiere oben
  zeigeweiter
  gleichspringe unten
springe oben
markiere unten
```

Die vier Glieder des Richtungskreises zwischen 'südost' und 'nord' werden angezeigt. Die Programmzeile 'zeige nordost' bewirkt beim Richtungskreis, dass die Darstellung der Nord-Richtung ins Register 1 kommt. Von dort wird sie mit dem Kopiere-Befehl ins Register 2 gebracht, um später mit dem Gleichspringe-Befehl auf diese Richtung reagieren zu können.

5.4.3 Kommentare und Modelle

! <zeichenkette>

Das Ausrufezeichen am Beginn einer Programmzeile bewirkt, dass die gesamte Zeile als Kommentar verstanden wird und vom Programmeditor also übersprungen wird.

Beispiel:

```
! Dies ist ein Kommentar.
var i
merke i = *01001001011000101
! zeige i
```

Der Wert der Variablen i wird nicht angezeigt, da die Programmzeile mit dem Zeige-Befehl als Kommentar gekennzeichnet wurde.

:: <modellname>

Möchte man sicher sein, dass ein Programm in dem Modell ausgeführt wird, für das es gedacht ist, stellt man diesen Befehl in den Programmtext. <modellname>.txt muss eine Datei bezeichnen, die die Modelldaten enthält.[112]

Beispiel:

```
::tmodell1
vorwaerts
vorwaerts
links
```

Durch den doppelten Doppelpunkt wird dem Programmeditor bekannt gegeben, dass er vor dem Start der Simulation erst in das Modell wechseln muss, welches durch die Datei 'tmodell1.txt' beschrieben wird.

[112] Weiteres zu Modellen findet sich im Kap. 6.

5.5 Sequenzen

Abfolgen von Befehlen werden in der gewünschten Reihenfolge von oben nach unten in den Programmeditor eingetragen und in dieser Reihenfolge auch abgearbeitet. Die Reihenfolge ist nicht beliebig, wie schon die Programmtexte

```
var i
var j
merke i = $FEEE
merke j = $BABA
zeige i
zeige j
hole i
```

und nach Tausch von Zeile 5 und 6

```
var i
var j
merke i = $FEEE
merke j = $BABA
zeige j
zeige i
hole i
```

zeigen. Im ersten Programm steht in der Variablen i letztlich der Wert von j. Im zweiten Programm ändert i seinen Wert nicht. Sequenzen können so viele Programmzeilen umfassen, bis sich die Programmzeilen nicht mehr eindeutig durch die VIDAS -Maschine ansprechen lassen. Der Programmeditor[113] überwacht diese Bedingung und gibt im Fehlerfalle eine entsprechende Meldung aus.[114]

5.6 Auswahlen

An Auswahlen seien die einseitige, die zweiseitige und die mehrseitige zu unterscheiden. In Pascal-artigen Programmiersprachen werden diese dem Programmierenden durch if-then-, if-then-else- und case-Konstrukte ermöglicht.

5.6.1 Einseitige Auswahl

Eine Anweisung der Gestalt 'if a = b then write(a);' lässt sich über den Befehlssatz der VIDAS -Maschine ausdrücken durch:

[113] s. Kap. 5.8

[114] Wenn die Anzahl der Zeilen der Abfolgematrix A der VIDAS -Maschine n beträgt, können also auch nur höchstens n Programmzeilen zu einer Sequenz gehören.

```
zeige a
kopiere
zeige b
gleichspringe weiter
springe schluss
markiere weiter
zeige a
markiere schluss
```

Dabei seien 'a', 'b' zwei vorher vereinbarte Variablen.

5.6.2 Zweiseitige Auswahl

Eine Anweisung der Gestalt 'if a = b then write(a) else write('ungleich');' wird in der hier beschriebenen Programmierung nachgebildet durch:

```
merke meldung = "ungleich
zeige a
kopiere
zeige b
gleichspringe weiter
zeige meldung
springe schluss
markiere weiter
zeige a
markiere schluss
```

Dabei seien 'a', 'b' und 'meldung' drei vorher vereinbarte Variablen.

5.6.3 Mehrseitige Auswahl

Eine Anweisung der Gestalt 'case a of 1: write('ungerade'); 2: write('gerade'); 3: write('ungerade'); ... else write('undefiniert');' ist ebenfalls durch den VIDAS -Befehlssatz möglich:

```
merke meldung_ug = "ungerade
merke meldung_g  = "gerade
merke meldung_ud = "undefiniert

merke b = $1
zeige b
kopiere
```

```
zeige a
gleichspringe weiter1
springe weiter2
markiere weiter1
zeige meldung_ug
springe schluss

markiere weiter2
merke b = $2
zeige b
kopiere
zeige a
gleichspringe weiter3
springe weiter4
markiere weiter3
zeige meldung_g
springe schluss

markiere weiter4
merke b = $3
zeige b
kopiere
zeige a
gleichspringe weiter5
springe weiter6
markiere weiter5
zeige meldung_ug
springe schluss

...

markiere weiter_sonst
zeige meldung_ud

markiere schluss
```

5.7 Wiederholungen

An Schleifen-Strukturen (Wiederholungen) bieten Pascal-artige Programmiersprachen for-Schleifen, abweisende und nichtabweisende Schleifen. Bei der for-

Schleife ist die Anzahl der Wiederholungen des Schleifenkörpers von keiner Bedingung abhängig. Bei der abweisenden Schleife wird *vor* jedem Durchlauf des Schleifenkörpers eine Laufbedingung überprüft. Bei der nichtabweisenden Schleife steht die Überprüfung einer Abbruchbedingung am Ende des Schleifenkörpers.

5.7.1 for-Schleifen

Möchte man eine Schleife der Gestalt 'for i := 1 to 10 do write('Hallo Welt!');' auf der VIDAS -Maschine nachbilden, dann gibt man das wie folgt ein:

```
var i
var meldung
merke meldung = "Hallo Welt!
&zfolge 10
zeige null
hole i
markiere oben
  zeige meldung
  zeige i
  zeigeweiter
  hole i
  nullspringe unten
springe oben
markiere unten
```

Nach dem Erzeugen einer Zahlenfolge wird durch die Programmzeile 'zeige null' die Darstellung von 'eins' ins Register 1 gebracht und dann als Wert in die Variable i geholt. Der Schleifenkörper zwischen 'markiere oben' und 'springe oben' wird dank des Zeigeweiter-Befehls genau zehn Mal durchlaufen.

5.7.2 abweisende Schleifen

Eine Schleife der Gestalt 'while a <> 3 do begin inc(a); write(a) end;' findet seine Umsetzung durch:

```
var a
&zfolge 10
zeige null
hole a
zeige zwei
kopiere
markiere oben
```

```
    zeige a
    gleichspringe unten
    zeigeweiter
    hole a
  springe oben
  markiere unten
```

Durch 'zeige null' wird der Wert von 'eins' ins Register 1 geschrieben, von wo er in die Variable a gelangt. Durch 'zeige zwei' kommt der Wert von 'drei' ins Register 1 und durch den Kopiere-Befehl ins Register 2. Am Beginn des Schleifenkörpers wird der Wert von a angezeigt und über den Gleichspringe-Befehl mit der 'drei' verglichen. Bei Ungleichheit läuft die Schleife weiter.

5.7.3 nichtabweisende Schleifen

Um eine Schleife der Gestalt 'repeat inc(a); write(a) until a = 3;' abzuarbeiten, lässt sich das so ansetzen:

```
var a
&zfolge 10
zeige null
hole a
zeige zwei
kopiere
markiere oben
  zeige a
  zeigeweiter
  hole a
  gleichspringe unten
springe oben
markiere unten
```

Erst am Ende des Schleifenkörpers wird auf Gleichheit geprüft und die Schleife gegebenfalls abgebrochen.

5.8 Programmeditor 4.9

Die auf der Grundlage dieses Kapitels 5 möglichen Programme werden durch die im Kapitel 4.9 beschriebenen VIDAS -Maschinen robust gespeichert und ausgeführt. Sie werden in der Hardware-Simulation über den Digitalsimulator ProfiLab geladen und dann durch die Maschine abgearbeitet. Für einen Umgang mit größeren Matrizen reichen auf Seiten des Wirtsrechners die Resourcen, die für eine

solche Hardware-Simulation nötig wären, jedoch nicht aus. Um sich aber nun doch in einer Größenordnung von 10^6 bis 10^7 Synapsen pro Matrix bewegen zu können, wurde die ProfiLab-Hardware-Simulation wiederum softwaremäßig simuliert. Dazu entstand ein Programmeditor 4.9, der im Unterschied zum Editor in Kap. 4.11 nicht die Belegung der Matrizen generiert und geeignet in Dateien der Hardware-Simulation zur Verfügung stellt, sondern der die Simulation selbst ausführt. Somit stellt er eine Entwicklungsumgebung für die Assoziative Programmierung dar, angereichert mit Möglichkeiten zur Veranschaulichung der Vorgänge in den Matrizen des Systems.

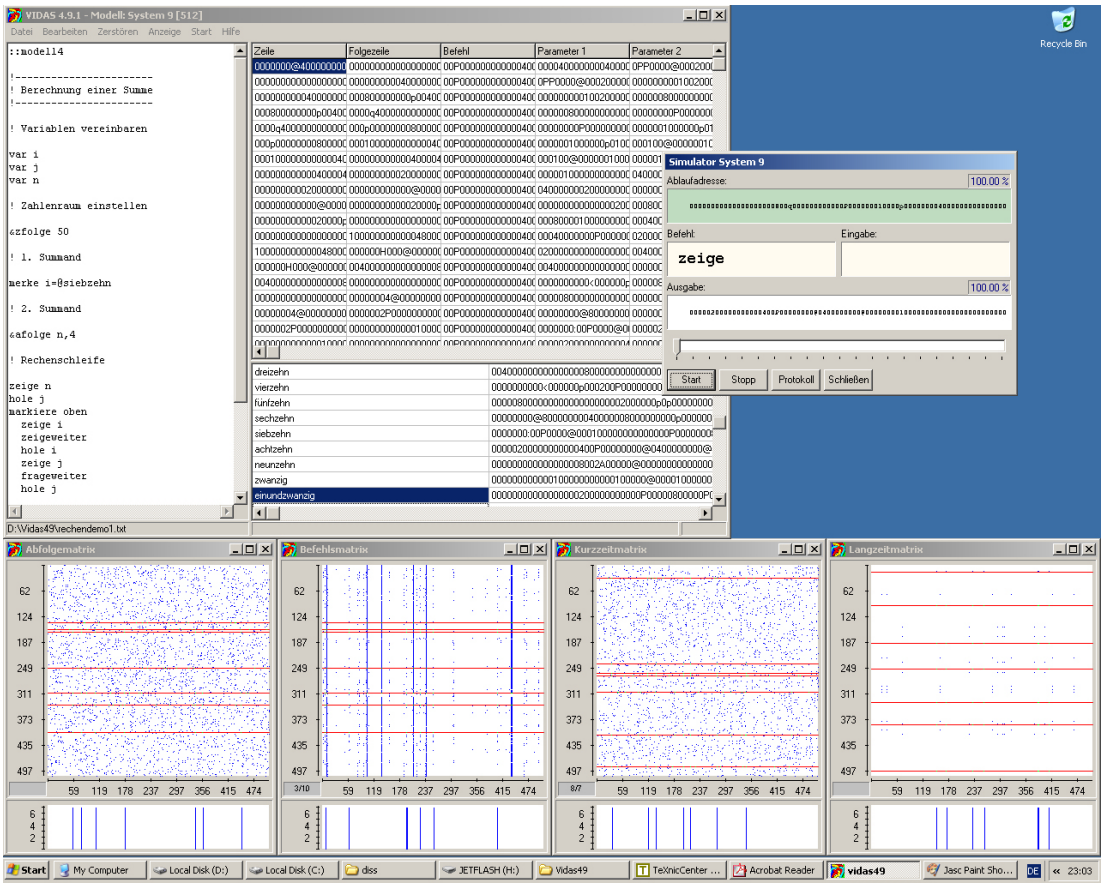


Abb. 70: Programmeditor 4.9

Nach dem Start des Programmeditors 4.9 erscheint das bereits in Kap. 4.11 beschriebene Editorfenster mit den Anzeigen der Belegungen der Matrizen und der Zuordnungstabelle für die Variablen (s. Abb. 70, oben). Durch Mausklick auf die Spaltenüberschriften dieser Anzeigen, werden die Spalten breiter angezeigt. In das eigentliche Editorfenster (ganz links) trägt man den Programmtext ein. Eine

Menüzeile bietet die Auswahl zwischen Dateioperationen ('Datei'), Editorarbeiten ('Bearbeiten'), Möglichkeiten zur Robustheitsprüfung durch Zerstörung der Matrizen ('Zerstören'), Anzeigeoptionen ('Anzeige'), Start der Simulation ('Start') und Hilfestellungen ('Hilfe'). Das Editorfenster kann durch die Maus auf beliebige Größe gebracht werden, falls die Anzeigefenster zu klein erscheinen.

Um einen Simulatorlauf zu beginnen, wählt man im Menü den Punkt 'Start' aus oder drückt auf die Taste F9. Es erscheint ein Anzeigefenster (s. Abb. 70, oben rechts), welches der Frontplatte in der ProfiLab-Simulation entsprechen soll,[115] jedoch um einige Möglichkeiten zur Fehlersuche und Ablaufsteuerung ergänzt wurde (s. Abb. 71). Durch drei Bedienknöpfe kann der Simulatorlauf gestoppt, gestartet oder durch ein Protokoll begleitet werden. Ein vierter Knopf sorgt gegebenenfalls für das Schließen des Fensters. Oberhalb der Bedienknöpfe befindet sich der Schieberegler für die Simulationsgeschwindigkeit (nach links — schneller, nach rechts — langsamer). Im Feld 'Ablaufadresse' taucht die Bitfolge auf, die die aktuelle Programmzeile repräsentiert. Im Feld 'Befehl' sieht man den aktuell ausgeführten Befehl, daneben eine eventuelle Eingabe an das Register 2. Diese Eingabe wird in der derzeitigen Implementation über die fünf Eingabeleitungen der Druckerschnittstelle des Wirtsrechners erwartet. Das 'Ausgabe'-Feld gibt den Inhalt von Register 1 wieder.

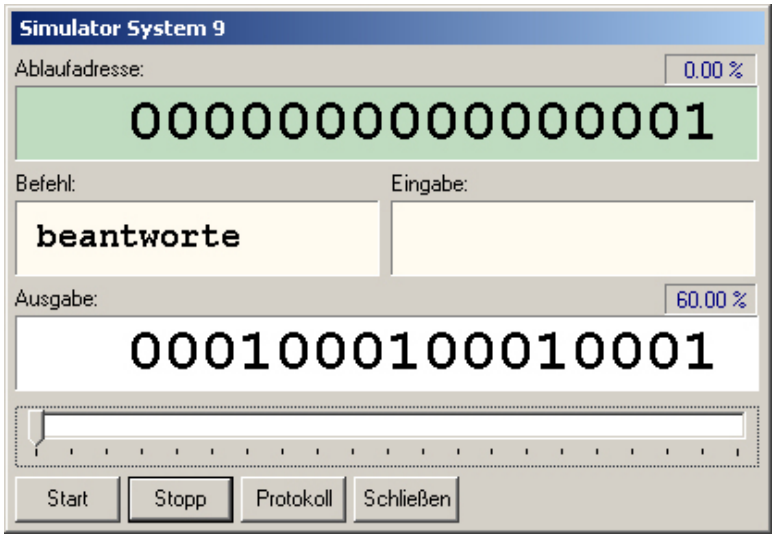


Abb. 71: Ausgabeeinheit des Programmeditors 4.9

Rechts oberhalb der Felder 'Ablaufadresse' und 'Ausgabe' wird die Schwellwertgüte in Prozent angezeigt. Das Programm stoppt, wenn sich hier für die Ablaufadresse nahezu 0 Prozent einstellen, da zur letzten Programmzeile keine Folgezeile

[115] s. z.B. Abb. 59

gelernt wird. Ein Wert kleiner als 100 Prozent für die Ausgabe bedeutet, dass es bei der Abfrage Störungen gab. Entweder wurde die Antwort auf eine Frage verlangt, die nicht gelernt worden ist, oder es wurden synaptische Verbindungen in der Matrix verändert. Letzteres wird zum Testen der Störunanfälligkeit über den Menüpunkt 'Zerstörungen' absichtlich herbeigeführt. Falls die Bitfolge in der 'Ausgabe' zu einer Variablen gehört, hilft ein Klick mit der Maus in dieses Feld, um die Variable in der Variablenliste des Programmeditors zu finden.

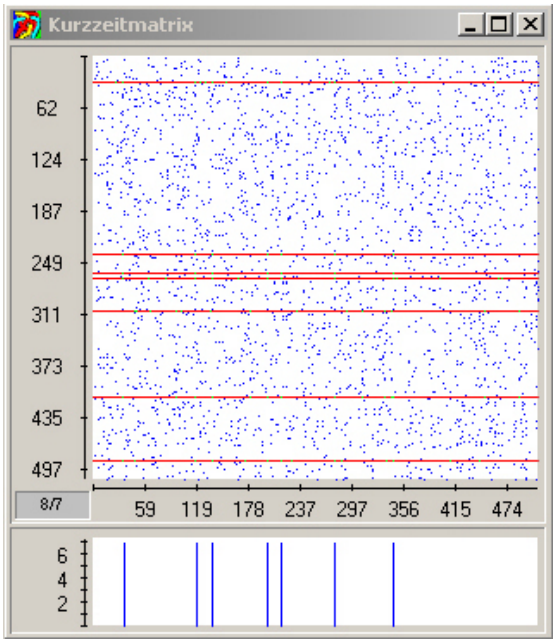


Abb. 72: Matrixanzeigefenster

Für einen Einblick in die Belegung einer Matrix und um in ihr gegebenenfalls Veränderungen vornehmen zu können, stehen nach Druck auf die Tasten Strg-F8 oder über den Menüpunkt 'Anzeige' eine Reihe von Fenstern zur Verfügung (s. Abb. 72), die man sich in beliebiger Größe anzeigen lassen kann. Zur Laufzeit der Simulation werden auf Wunsch die jeweils aktiven Matrixzeilen farbig markiert. Die Koordinaten der Synapsen, bei denen sich der Mauszeiger gerade befindet/befand, erscheinen in einem kleinen Feld links der Rechtsachse. Oberhalb der Rechtsachse wird die Matrix dargestellt, unterhalb der Rechtsachse tauchen die Schwellwerte der letzten Abfrage auf. Mit einem Mausklick in das Feld mit der Matrixanzeige erscheint eine Vergrößerung des Teils der Matrix, in welchem sich der Mauszeiger während des Klicks befand (s. Abb. 73).

In diesen Vergrößerungsfenstern erreicht man mittels Mausklicks mit der linken Maustaste ein Umschalten der unter dem Mauszeiger befindlichen Synapse. Mit

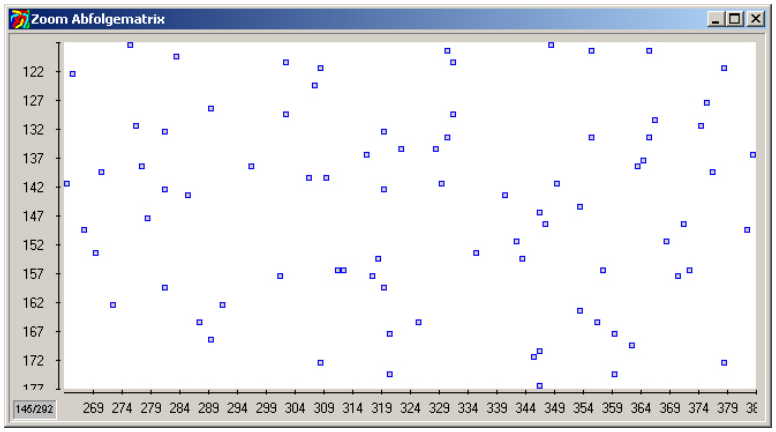


Abb. 73: Vergrößerung eines Teils der Abfolgematrix

der rechten Maustaste erhält man über ein Submenü die Möglichkeit, verschiedene Zerstöraktionen im Matrixausschnitt vornehmen zu können (s. Abb. 74).

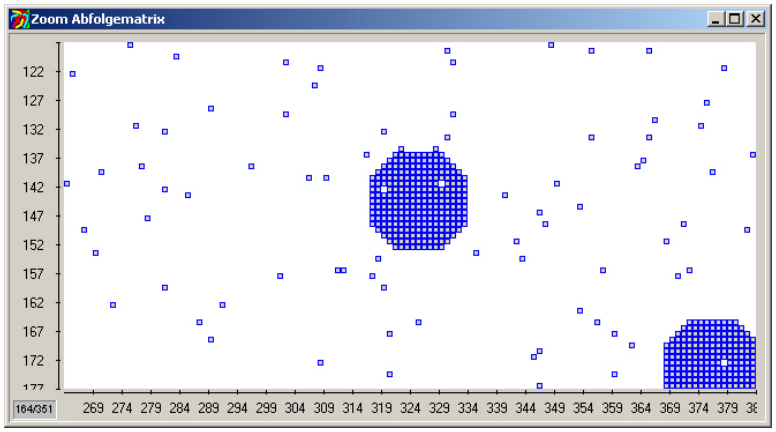


Abb. 74: Zerstörung eines Teils der Matrix

Solche Zerstörungen werden auch sogleich in der Matrixanzeige sichtbar. Mit einem erneuten Simulationslauf wird danach geprüft, ob das Programm trotz dieser Eingriffe noch korrekt abläuft.

Für das dem Langzeitgedächtnis L nebengeordnete ROM startet eine Eingabehilfe nach Drücken von F8 oder über Auswahl von 'Anzeige' (s. Abb. 75).

Bei jedem Mausklick auf eines der durch die ROM-Anzeige mit '0' und '1' dargestellten Bits, wechselt dieses seinen Zustand. Die drei Schaltknöpfe 'Löschen', 'Schließen' und 'Speichern' erlauben das Löschen des ROMs, das Beenden der Bearbeitung des ROMs (ohne zu speichern) und das Abspeichern der eingestellten

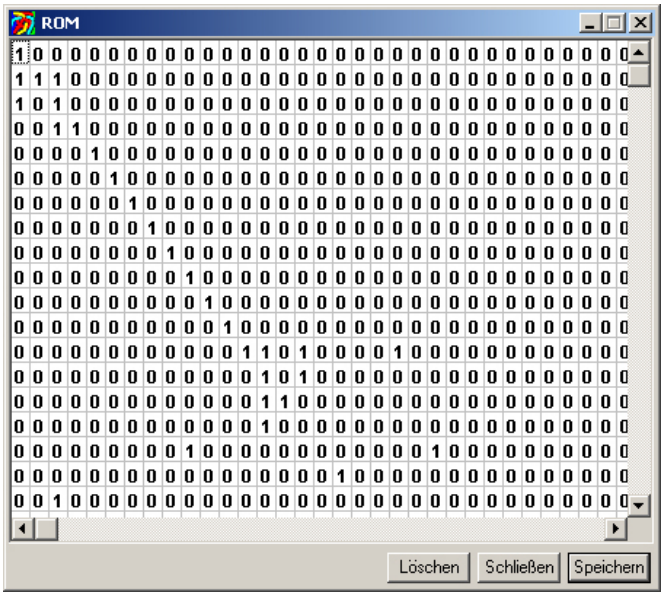


Abb. 75: Eingaben ins ROM

Daten. Das ROM passt sich in der Größe stets der Größe des Langzeitgedächtnisses L an.

Möchte man Eingaben über die fünf Eingabeleitungen der Druckerschnittstelle[116] entgegennehmen, schaltet man diese vorher über den Menüpunkt 'Anzeige' ein. Über den gleichen Menüpunkt erreicht man auch, dass die Simulation in Einzelschritten abgearbeitet wird. Zwischen je zwei Programmzeilen stoppt der Simulatorlauf und wird erst nach einem Mausklick fortgesetzt.

Damit der Programmeditor 4.9 nicht nur eine VidAs -Maschine des Zuschnitts simuliert, der in Kapitel 4.9 beschrieben wurde, unterstützt der Editor ein Modellkonzept, durch das sich Erweiterungen der Maschine beschreiben lassen. Das folgende Kapitel 6 erklärt dieses. Die jeweils gewünschte Modelldatei kann man über den Menüpunkt 'Datei' laden oder durch einen '::'-Befehl[117] in den Programmtext hineinschreiben. Bevor nach einem Modellwechsel das nächste Programm ausgeführt wird, stellt der Programmeditor alle Matrixgrößen, Schwellwerttupel, Fenster, Darstellungsweisen, Kodierungen und Befehlssätze auf das gewählte Modell ein.

[116] Dieses sind die Leitungen BSY, ACK, PE, SLT und ERR des parallelen Portbausteins.
[117] s. Kap. 5.4.3

6 VidAs -Modelle

Beim Übergang von VidAs 7 auf VidAs 9 [118] wurde die Matrixgröße im Hardwaresimulator von 8 x 8 auf 16 x 16 Synapsen pro Matrix erweitert, also von gut 300 auf gut 1.500 zu simulierende Flip-Flops. Die Startzeit der Simulation durch ProfiLab stieg dadurch von gut 3 auf 12 Minuten. Eine nochmalige Verdoppelung der Matrixbreite würde die Startzeit vermutlich auf über eine Stunde erhöhen; ProfiLab oder der Wirtsrechner stoßen hier an ihre Grenzen.

Um dennoch Erkenntnisse über das Verhalten einer VidAs -Maschine der Art VidAs 9 zu bekommen, wenn die Matrixgröße steigt, wurde der Programmeditor 4.9[119] mit einem Modellkonzept versehen. Dieses erlaubt, dem Editor durch eine Beschreibungsdatei (Modelldatei) mitzuteilen, auf welche Matrixgröße, welches Anzeigealphabet, welche Befehle und welche Bitdichte er sich einstellen soll.

Beispiel einer Modelldatei:

```
name=System 9 [16]
matsize=16
bitdichte=3
dispalphabet=01
$merke         = 0000000100000001,V,ZK
$lerne         = 1000000000000001,VK,ZK
$lies          = 0000001000000010,V,
$zeige         = 0000010000000100,V,
$beantworte    = 1000000000000100,VK,
$hole          = 0000100000001000,V,
$nullspringe   = 0001000000010000,M,
$zeigeweiter   = 0010000000100000,,
$gleichspringe = 0100000001000000,M,
$frageweiter   = 0100000000100000,,
$laderom       = 0100000010000000,,
$kopiere       = 1010000000000000,,
$pausiere      = 0000000000000000,,
```

Mit dieser Modelldatei wird genau die Maschine VidAs 9 beschrieben. Die Reihenfolge der Zeilen in der Modelldatei ist beliebig. Hinter dem Bezeichner 'name' wird ein Name für dieses Modell angegeben. Dieser Name taucht dann im Titelbalken des Editorfensters auf, um das gewählte Modell zu nennen. Hinter 'matsize' steht die Matrixgröße (hier: 16 x 16 Synapsen). Mit 'bitdichte' (in Prozent) nimmt

[118] vgl. Kap. 4.9
[119] vgl. Kap. 5.8

man Einfluss auf die Anzahl der Einsen, die beim Kodieren zu verteilen sind.[120] [121] Auf 'dispalphabet' folgen die Zeichen, mit denen die Ausgaben im Ausgabefenster[122] vorgenommen werden sollen. Hier sind es nur die beiden Zeichen '0' und '1', also erfolgt die Ausgabe binär.

Die Zeilen, die mit dem Zeichen '$' eingeleitet werden, legen für einen Befehl fest, unter welchem Code er in die Befehlsmatrix B eingetragen werden soll. An den Code schließt sich die Angabe der Parametertypen an, die für den ersten und zweiten Parameter zugelassen sind. Ein 'V' bedeutet, dass an der Position des ersten Parameters eine Variable stehen kann, 'K' steht für eine Konstante, 'Z' für einen Pointer, 'M' für eine Marke. Ist bei einem Befehl nur ein oder kein Parameter zugelassen, bleibt die Aufzählung leer, die Kommata sind aber zu setzen.

Möchte man es dem Programmeditor 4.9 überlassen, einen Code für die gewünschten Befehle auszuwählen, gibt man statt einer Bitfolge das Zeichen '%' ein.

Erhöht man in dieser Modelldatei die Matrixgröße mit 'matsize = 256', so geht der Editor mit Matrizen um, die 2^{16} Synapsen besitzen. Es empfiehlt sich dann aber, auch das Anzeigealphabet zu verlängern, damit die Anzeigen besser lesbar bleiben:

```
name=System 9 [256]
matsize=256
bitdichte=3
dispalphabet=0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijkl
```

Um ein Alphabet von 128 oder 256 Zeichen Länge vom Editor automatisch erzeugen zu lassen, ist auch folgende Kurz-Eingabe möglich:

```
dispalphabet=<ascii128>
```

beziehungsweise

```
dispalphabet=<ascii256>
```

Wenn in einem Modell neue Befehle eingeführt werden sollen, die den Befehlsvorrat[123] von VIDAS 9 erweitern, dann ist für den Editor zusätzlich ein Programmmodul zu schreiben, um die neuen Befehle wunschgemäß ausführen zu können. In den Kapiteln 6.2 bis 6.4 werden drei solcher Modelle vorgestellt.

[120] Zum Beispiel erfährt der Programmeditor so, wie viele Einsen er einer Variablen in ihrer binären Darstellung geben soll.
[121] Zu Kodierungen s. Kap. 8
[122] s. Abb. 71
[123] s. Kap. 5.3

6.1 System 9

Alle Modelle, die mit dem Befehlsvorrat von VIDAS 9 auskommen, zählen zum „System 9“ gehörig. Zur Laufzeit des Programmeditors 4.9 wird das zugehörige Modul benutzt. In der Modelldatei muss hinter dem Eintrag 'name' der Bezeichner 'System 9' folgen, um dem Editor die Wahl genau dieses Moduls mitzuteilen.

Mit dem System 9 lässt sich schon bei einer Matrixgröße von 512 x 512 Synapsen ein Algorithmus zum Addieren zweier Zahlen angeben, wie Kap. 7.1 vertiefen wird. Dazu wird an den Beginn der Modelldatei eingetragen:

```
name=System 9 [512]
matsize=512
bitdichte=1.3
dispalphabet=<ascii128>
```

6.2 Turtle-Grafik

Das Modell „Turtle“ enthält zusätzlich zum „System 9“ vier Bewegungs-Befehle für eine gedachte Turtle. Die Idee einer Turtle-Grafik findet man in der Programmierumgebung von LOGO.[124] Die Modelldatei enthält am Anfang die Zeilen:

```
name=Turtle [1024]
matsize=1024
bitdichte=1
dispalphabet=<ascii128>
$vorwaerts     = %,,
$rueckwaerts   = %,,
$rechts        = %,,
$links         = %,,
```

Mit den beiden Befehlen 'vorwaerts' und 'rueckwaerts' wird die Turtle einen Schritt auf einer gedachten Zeichenfläche vorwärts bzw. rückwärts geschickt und sie hinterlässt dabei ihre Spur. 'rechts' und 'links' bewirken Drehungen der Turtle um 45 Grad. Mit dem Einsatz einer Turtlegrafik verbindet man in LOGO die Absicht, durch das planvolle Erzeugen geometrischer Figuren Schülern den Einstieg ins Denken eines Programmierenden zu ebnen.

6.3 Robot

Das Modell „Robot“ enthält zusätzlich zu „System 9“ Befehle zur Wahrnehmung einer Umgebung und zum Lernen eines Weges durch einen gedachten „Wald“ aus

[124] vgl. [Abelson 83]

Merkmalen. Der Anfang der zugehörigen Modelldatei nennt die nötigen Einstellungen:

```
name=Robot [1024](30)
matsize=1024
bitdichte=1
dispalphabet=<ascii256>
merkmale=30
$schaue        = %,V,
$gehezumstart  = %,,
$folgedermaus  = %,,
$erkenne       = %,,
$gehe          = %,,
$farbwechsel   = %,,
```

Die Anzahl der Merkmale wird in der Modelldatei durch einen Eintrag hinter 'merkmale' festgelegt. Fehlt der Eintrag, werden dreißig Merkmale angenommen. Durch den Befehl 'schaue' legt der Robot die wahrgenommene Umgebung (eine Bitfolge in Matrixbreite) in der Variablen ab, deren Bezeichner an erster Parameterposition steht. Wenn die Umgebung „leer“ ist, erhält die Variable als Wert eine Bitfolge, in der nur Nullen stehen. Mit 'gehezumstart' kehrt der Robot an einen Startpunkt in der Mitte des „Waldes“ zurück. Über 'folgedermaus' wird die Lernphase des Robots verwirklicht. Er legt seinen Umgebungseindruck im Langzeitgedächtnis L ab und assoziiert mit ihm die Gehrichtung, die dem Robot über den Mauszeiger angegeben wird.

Der Befehl 'gehe' lässt den Robot einen Schritt in die Richtung vorwärts gehen, die durch das Register 1 angezeigt wird. Den Wert für Register 1 liefert die Anweisung 'erkenne', die die Matrix L mit dem Umgebungseindruck abfragt und daraufhin von dieser eine Richtung genannt bekommt. Findet 'gehe' in Register 1 nur den Wert „0“, macht er einen Schritt in eine zufällig gewählte Richtung. Dank eines verlangten 'farbwechsel's lässt sich der Weg des Robots in Rot oder Blau verfolgen.

6.4 Homunkulus

Als Variante des „Robot“-Modells bietet sich das Modell „Homunkulus“ an. In diesem soll eine gedachte Figur (Homunkulus[125]) in die Lage versetzt werden, sich in einem Merkmalswald zu bewegen, ohne vorher einen Weg gelernt zu haben. Dazu soll sich der Homunkulus merken, wo er bereits einmal gewesen ist,

[125]Mephistopheles: „Fürwahr, es ist Homunculus! Woher des Wegs, du Kleingeselle?“ Homunculus: „Ich schwebe so von Stell' zu Stelle [...]“, aus Goethe: „Faust, 2. Teil“, 2. Akt

um beim selbsttätigen Herumlaufen nicht in eine Endlosschleife zu geraten. Der Modelldateianfang enthält folgende Einträge:

```
name=Homunkulus [2200](45)
matsize=2200
bitdichte=1
dispalphabet=<ascii256>
merkmale=45
$schaue                  = %,V,
$erinnere                = %,,
$pruefefeld              = %,,
$macheschrittvorwaerts   = %,,
$macheschrittrueckwaerts = %,,
```

Mit 'schaue' wird wie im „Robot"-Modell der Umgebungseindruck in einer Variablen abgelegt. Durch 'erinnere' wird dieser Umgebungseindruck ins Langzeitgedächtnis L eingetragen. Der Befehl 'pruefefeld' legt den Wert desjenigen Merkmals in Register 1, welches das Feld besitzt, das der Homunkulus als nächstes betreten würde, wenn sich seine Laufrichtung nicht ändert. Durch 'macheschrittvorwaerts' bzw. 'macheschrittrueckwaerts' geht der Homunkulus in der aktuellen Laufrichtung vor und zurück.

Die in den obenstehenden Modellen genannten zusätzlichen Befehle wären in eine VIDAS 9-Hardware mit keinem großen Aufwand einzubauen, indem man sie entweder gleich in der Maschine entschlüsseln lässt und nur die notwendigen Leitungen nach außen führt (Ansteuerung von Relais, Schrittmotoren, Sensoren o.ä.), oder indem man alle Befehlsleitungen aus der VIDAS -Maschine von außen zugreifbar gestaltet, so dass die Entschlüsselung der Befehle und die sich daraus ergebenden Handlungen zur Gänze extern abgewickelt werden.

Das Kapitel 7 nennt Motive und Anwendungsbeispiele für die hier vorgestellten Modelle.

7 Anwendungen

7.1 Rechnen ohne Rechenwerk

Die VidAs -Maschine besitzt kein Rechenwerk. Die Matrizen müssen das Rechnen erst beigebracht bekommen. Zwar könnte man an die Befehlsleitungen, die von der Befehlsmatrix B ausgehen,[126] ein Rechenwerk anschließen und durch die VidAs -Maschine zum Rechnen bringen, doch das, was in diesem Rechenwerk dann abliefe, entzöge sich dem Wunsch nach Störunanfälligkeit der Maschine.

Eine Hilfe bei der Suche nach geeigneten Programmen, die die Maschine zum Rechnen bringt, ist die Vorstellung von dem, was ein Erstklässler beim Rechnen wohl täte. Womöglich hat er den Zahlenraum bis 50 kennengelernt und bekommt nun die Aufgabe vorgelegt, die Summe 17 + 4 zu berechnen. Vielleicht wird der Schüler mit den Fingern von 17 ausgehend um vier Finger weiterzählen.

Nachfolgendes Programm greift dieses Vorgehen auf, indem es eine Zahlenkette vereinbart,[127] bei 'siebzehn' zu zählen beginnt und dann in einer Schleife vier Mal den Nachfolger bestimmt.

```
::modell4

!-----------------------
! Berechnung einer Summe
!-----------------------

! Variablen vereinbaren

var i
var j
var n

! Zahlenraum einstellen

&zfolge 50

! 1. Summand

merke i=@siebzehn
```

[126] vgl. Abb. 62
[127] vgl. Kap. 5.4.2

```
! 2. Summand

&afolge n,4

! Rechenschleife

zeige n
hole j
markiere oben
  zeige i
  zeigeweiter
  hole i
  zeige j
  frageweiter
  hole j
  nullspringe unten
springe oben
markiere unten

! Ergebnisanzeige

zeige i
```

Das Ergebnis steht am Ende des Programmlaufs in Register 1. Alle Programme dieses Kapitels laufen in einem „System 9“-Modell.

Auch eine Multiplikation zweier Zahlen könnte auf der Grundlage derselben Idee durch eine geschachtelte Schleife berechnet werden, indem so oft wie nötig der Nachfolger in einer Zahlenkette bestimmt wird.

```
::modell5

!--------------------------
! Berechnung eines Produkts
!--------------------------

! Variablen vereinbaren

var i
var j
var k
var m
var n
```

```
! Zahlenraum einstellen

&zfolge 60

! 1. Faktor

&afolge m,17

! 2. Faktor

&afolge n,3

! Rechenschleifen

merke i=@null
zeige n
hole k
markiere ganzoben
  zeige m
  hole j
  markiere oben
    zeige i
    zeigeweiter
    hole i
    zeige j
    frageweiter
    hole j
    nullspringe unten
  springe oben
  markiere unten
  zeige k
  frageweiter
  hole k
  nullspringe ganzunten
springe ganzoben
markiere ganzunten

! Ergebnisanzeige
```

```
zeige i
```

Die Praxis zeigt, sei es beim Schüler oder bei diesem Programm, dass das Rechnen über Vorgänger- und Nachfolgerbildung umständlich und zeitaufwändig ist.

So wird der Schüler also sehr bald ein schriftliches Rechenverfahren anwenden, um die Summen von Zahlen zu bilden, bei dem er ein ziffernweises Vorgehen und die Verwaltung eines Übertrags erlernt. Dasselbe Problem lösten bereits Wilhelm SCHICKARD und Blaise PASCAL, indem sie in ihren Rechenmaschinen dafür sorgten, dass zehnstufige Zahnräder durch eine Klaue und Mitnehmerstifte das höherwertige Zahnrad um eine Stelle weiterdrehten (s. Abb. 76).

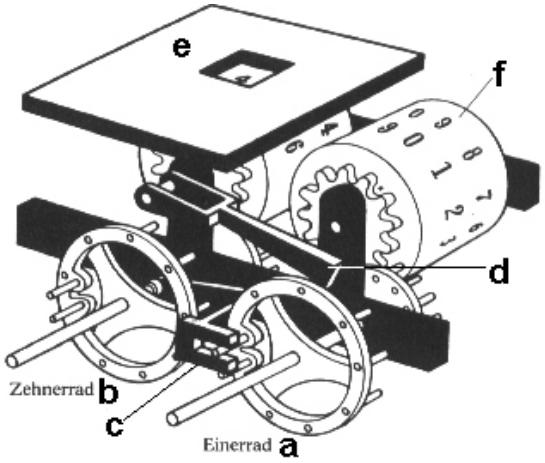


Abb. 76: Zahnräder von Pascaline, der Rechenmaschine von Blaise PASCAL

An die Stelle der Zahnräder so einer mechanischen Rechenmaschine kann ein Ziffernkreis aus der Assoziativen Programmierung treten.[128] Der folgende Algorithmus leistet die Berechnung der Summe zweistelliger Zahlen mit Hilfe dieser Art von „Zählrädern“.

```
::modell4

!----------------------------------------
! Berechnung der Summe 2-stelliger Zahlen
!----------------------------------------

! Variablen vereinbaren

var i0
```

[128] vgl. Kap. 5.4.2

```
var i1
var j
var n0
var n1
var übertrag
var tmp

! Ziffernkreis 0-9 einrichten

&zkreis

! Null merken

merke tmp=@null
zeige tmp
kopiere

! 1. Summand

merke i0=@fünf
merke i1=@sieben

! 2. Summand

&afolge n0,7
&afolge n1,2

! Rechenschleife Einer

merke übertrag=@null
zeige n0
hole j
nullspringe unten1
markiere oben1
  zeige i0
  zeigeweiter
  hole i0
  gleichspringe übertragsetzen1
  markiere weiter1
  zeige j
```

```
  frageweiter
  hole j
  nullspringe unten1
springe oben1
markiere unten1

! Übertrag verrechnen

zeige übertrag
gleichspringe prossimo1
zeige i1
zeigeweiter
hole i1
gleichspringe prossimo2
markiere prossimo1
merke übertrag=@null
springe prossimo3
markiere prossimo2
merke übertrag=@eins
markiere prossimo3

! Rechenschleife Zehner

zeige n1
hole j
nullspringe unten2
markiere oben2
  zeige i1
  zeigeweiter
  hole i1
  gleichspringe übertragsetzen2
  markiere weiter2
  zeige j
  frageweiter
  hole j
  nullspringe unten2
springe oben2
markiere unten2

! Ergebnisanzeige
```

```
zeige i0
zeige i1
zeige übertrag

springe schluss

! Unterprogramme Übertrag

markiere übertragsetzen1
  merke übertrag=@eins
springe weiter1

markiere übertragsetzen2
  merke übertrag=@eins
springe weiter2

markiere schluss
```

Wenn während des Drehens eines Ziffernkreises die 'null' auftaucht, springt der Programmablauf in ein Unterprogramm, welches der Übertragsvariablen 'übertrag' eine 'eins' zuweist. Das entspricht der oben genannten Klaue, die den Überlauf dem nächsten Zählrad mitteilt. Die Abarbeitung dieses Programms geschieht im Mittel deutlich schneller als das Programm, welches eingangs vorgestellt wurde. Die Erweiterung auf die Addition n-stelliger Zahlen wäre einfach, da sich der Ablauf für jede Stelle in gleicher Weise wiederholt. Auch könnte man statt 10er-Zählrädern welche nehmen, die zum Beispiel 16 Zähne haben, um im Hexadezimalsystem zu rechnen. Eine weitere Variante zum Berechnen von Produkten mit Hilfe eines Bündels von Assoziativmatrizen wurde in [Dierks 88a] angegeben. Je eine Matrix leistete die Multiplikation mit einer einstelligen Zahl. Der Zahlenraum blieb dadurch allerdings allzu eingeengt.

7.2 Störunanfälligkeit

Geometrische Figuren wie sie mit einer Turtle-Grafik erzeugt werden, eignen sich gut, um vor Augen zu führen, dass Programme dank der fehlertoleranten Eigenschaften von Assoziativmatrizen trotz Störungen korrekt abgearbeitet werden. Dazu ist lediglich die geplante Figur mit der tatsächlich entstandenen Figur zu vergleichen. Das folgende Programm zeichnet eine Blütenfigur auf den Bildschirm.

Es wird im „Turtle“-Modell ausgeführt.[129]

```
::tmodell1

! Variablen vereinbaren

var kette1
var kette2
var hilf

! Assoziationsketten erzeugen

&afolge kette1,8
&afolge kette2,4

! Zeichnung

zeige kette2
hole hilf

markiere ganzoben
  zeige kette1
  markiere oben
    vorwaerts
    vorwaerts
    vorwaerts
    vorwaerts
    vorwaerts
    rechts
    frageweiter
    nullspringe unten
  springe oben
  markiere unten
  rechts
  rechts
  vorwaerts
  vorwaerts
  vorwaerts
  zeige hilf
```

[129] s. Kap. 6.2

```
    frageweiter
    hole hilf
    nullspringe ganzunten
  springe ganzoben
  markiere ganzunten
```

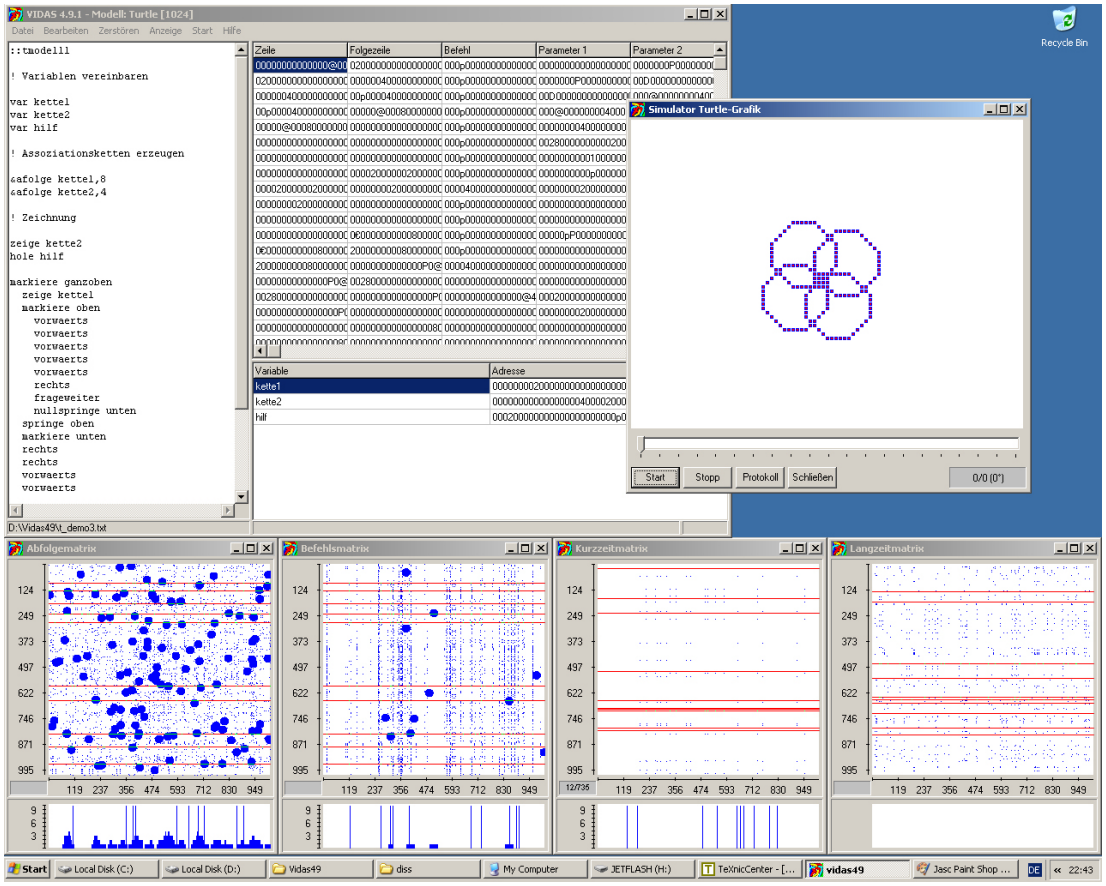


Abb. 77: Eine Figur wird trotz Störungen korrekt gezeichnet

Vor dem Start des Programms lässt man über den Menüpunkt 'Zerstören' den Programmeditor auf die Matrizen einwirken und sich diese anzeigen (s. Abb. 77). Durch die Darstellungen der Matrizen A, B, K und L wird veranschaulicht, wie sich Störungen auf die Schwellwerte auswirken (in Abb. 77 jeweils in den Fenstern ganz unten) und welche horizontalen Axone beim Programmlauf jeweils aktiv sind. Die Schwellwertmaxima setzen sich trotz der herbeigeführten Fehlfunktion vieler Synapsen klar gegen das „Rauschen“ durch (Fenster unten links). Auf eine Zerstörung der Art „XOR“ reagiert das System schneller mit Ausset-

zern, wie bereits die Ergebnisse von VIDAS 3 zeigten.[130] Doch dass dieser Fall in der Praxis auftritt (alle Bits einer gewissen Umgebung verkehren ihren Inhalt ins logische Gegenteil) gilt eher als unwahrscheinlich. Vielmehr zeigt die heutige Praxis, dass Bits bei Störungen bevorzugt den Wert „1“ annehmen, also der in Abb. 77 demonstrierte Fall eintritt.

Es gibt Autohersteller, die mit den durch elektromagnetische Einstrahlung ausgelösten Effekten[131] in der Steuerelektronik ihrer Fahrzeuge einige Probleme haben. Durch die Presse gingen Berichte von einem thailändischen Minister, der seinen Dienstwagen nicht mehr verlassen konnte,[132] von einem Fahrer, dessen Motor im Schweizer Gubrist-Tunnel zwischen Zürich und Bern wegen zu hoher Sendeleistung von Funkanlagen wiederholt ausfiel,[133] und von einem Fahrer, der seinen Wagen auf einer französischen Autobahn bei einer Geschwindigkeit von 200 $\frac{km}{h}$ weder abbremsen noch den Motor ausstellen konnte.[134] Der ADAC meldete zu Beginn des Jahres 2004, dass die Hälfte aller Pannen in Automobilen durch fehlerhafte Elektronik verursacht worden sei.[135] Es ist anzunehmen, dass der Einsatz einer fehlertoleranten Maschine wie oben vorgestellt in diesem Bereich Verbesserungen bewirken würde.

7.3 Pfadfindeproblem

In seinem Buch „Künstliche Wesen — Verhalten kybernetischer Vehikel“ schildert Valentin BRAITENBERG[136] Wesen mit Sensorenapparaten, die auf unterschiedliche Umweltfaktoren (z. B. Licht, Temperatur, Sauerstoffgehalt, Gehalt an organischem Material) reagieren. Dazu werden die Sensoren in mancherlei Weise mit den Motoren der Antriebsräder der Wesen verbunden. Das beobachtete Verhalten dieser Vehikel beschreibt der Betrachter womöglich mit Begriffen wie „Furcht“, „Agression“ bis „Egoismus und Optimismus“. Abb. 78 zeigt ein solches multisensorisches Vehikel. Die Menge dieser Vehikel bekomme im Folgenden Zuwachs.

Ein Vehikel, welches in seiner Umgebung eine gewisse Anzahl von Merkmalen wahrnehmen kann und welches die Verbindung zu seinen Antriebsrädern über eine

[130] vgl. Kap. 4.6.3 und Kap. 4.6.7

[131] EMV — Elektromagnetische Verträglichkeit, EMI — Electromagnetic Immunity, EMC — Electromagnetic Compatibility, vgl. http://www.triplec.co.uk/faqs.htm

[132] vgl. Jürgen NIEHAUS: „Fehler in elektronischen Steuergeräten“ vom 11. Februar 2004, http://www5.in.tum.de/~huckle/elektronikfehler.pdf

[133] vgl. http://www.oekosmos.de/article/articleview/266/1/16/

[134] vgl. http://www.spiegel.de/auto/aktuell/0,1518,347114,00.html vom 18. März 2005

[135] s. http://www5.in.tum.de/~huckle/elektronikfehler.pdf

[136] vgl. [Braitenberg 86, Teil 1]

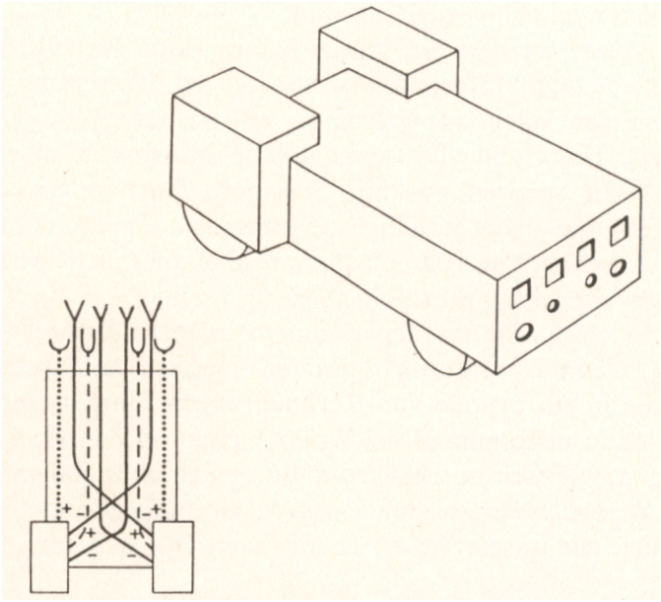


Abb. 78: Multisensorisches Vehikel nach Valentin BRAITENBERG

Assoziativmatrix herstellt, soll in einer Lernphase einen Weg lernen und diesem später selbst nach einigen Irritationen selbständig folgen können. Das Programm wird im „Robot“-Modell ausgeführt.[137]

```
::rmodell1

var i

! Starten und den Weg lernen

gehezumstart
markiere oben
  folgedermaus
  schaue i
  zeige i
  nullspringe unten
springe oben
markiere unten

! Zum Start zurück und dort "verirren"
```

[137] s. Kap. 6.3

```
gehezumstart
farbwechsel
merke i = 0
zeige i
kopiere
gehe
gehe
gehe
gehe
gehe

! Nach dem Weg suchen

markiere oben2
  erkenne
  gehe
  schaue i
  zeige i
  nullspringe unten2
springe oben2
markiere unten2
```

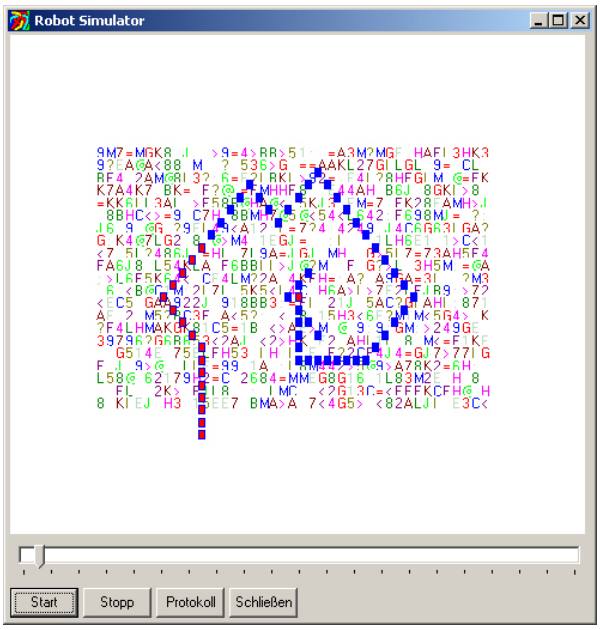


Abb. 79: Pfadfinder

Das Vehikel muss hier zwischen dem Lernen des Weges und der Suche des gelernten Pfades einige zufällige Schritte machen, indem die Richtung im Register 1 durch den Kopiere-Befehl gelöscht wurde und es daher in einer vom Zufallsgenerator bestimmten Richtung gehen muss. Dennoch findet es fast immer auf den gelernten Pfad zurück, wenn es diesem irgendwo begegnet, und folgt ihm dann bis zum Ende (s. Abb. 79).

Ändert sich der Merkmalswald zwischen Lern- und Laufzeit, kann man stets auf die dem System innewohnende Fehlertoleranz hoffen.

Zur Überprüfung des Verhaltens des Vehikels in teilweise veränderter Umgebung wurde der Befehl 'waldschaden' implementiert, der den Programmeditor veranlasst, 1 Prozent des Merkmalswaldes zufällig zu verändern. Bei zweimaligem Aufruf wären dann 2 Prozent per Zufall geändert und so fort. Der Programmtext ändert sich also wie folgt:

```
::rmodell2

var i

! Pfad lernen

gehezumstart
markiere oben
  folgedermaus
  schaue i
  zeige i
  nullspringe unten
springe oben
markiere unten

gehezumstart
farbwechsel
merke i = 0
zeige i
kopiere

! 1 Prozent des Merkmalswalds ändern

waldschaden

! Pfad suchen
```

```
markiere oben2
  erkenne
  gehe
  schaue i
  zeige i
  nullspringe unten2
springe oben2
markiere unten2
```

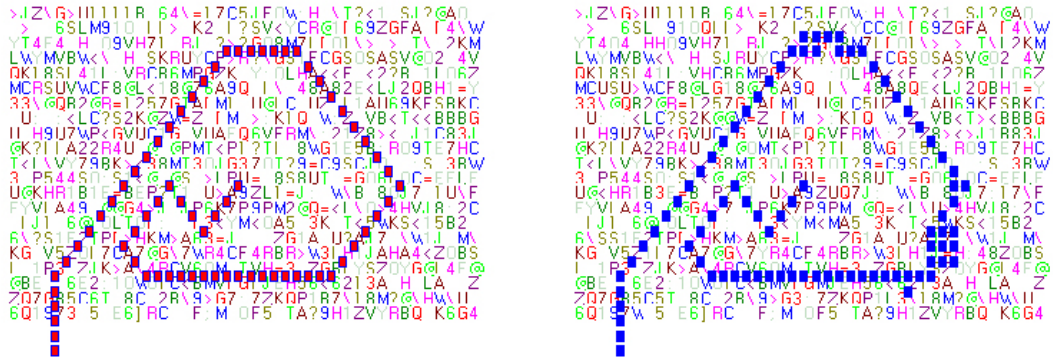


Abb. 80: Suchweg im veränderten Merkmalswald

Abbildung 80 gibt links den gelernten Pfad und rechts einen typischen Suchweg des Vehikels in einem solchen Wald wieder. Interessant ist, dass dieser zum Teil parallel zum gelernten Pfad verläuft, aber dann je nach Ausmaß der Veränderungen zumeist doch auf den gelernten Pfad zurückkehrt und zum Ausgang gelangt.

7.4 Irrwegeproblem

Vorgelegt sei ein Merkmalswald, in welchem sich eine gedachte Figur (Homunkulus) befindet, welche sich darin zu bewegen hat, ohne den Weg vorher gelernt zu haben und ohne endlos auf immergleichen Wegen herumzuirren. Eine Lernphase wie im Pfadfindeproblem in Kap. 7.3 gibt es hier also nicht. Statt dessen folgt der Homunkulus der einfachen Regel „Wenn ich an einen Ort komme, an dem ich schon einmal war oder an dem sich nichts befindet, bleibe ich stehen.“ Das folgende Programm führt diese Regel aus. Es wird im „Homunkulus“-Modell ausgeführt.[138]

[138] s. Kap. 6.4

```
::hmodell1

! Variablen für Richtung und Umgebung vereinbaren

var r
var u
var test

! Richtungskreis einrichten

&rkreis
zeige südost
hole r

! Ausgang suchen

schaue u
markiere ganzoben
  pausiere
  springe leertest
  markiere leertestzurueck
  zeige u
  erinnere
  zeige r
  kopiere
  markiere oben
    zeige r
    pruefefeld
    nullspringe weiter1
    zeige r
    zeigeweiter
    hole r
    gleichspringe schluss
  springe oben
  markiere weiter1
  zeige r
  macheschrittvorwaerts
  schaue u
  zeige u
  kopiere
```

```
  frageweiter
  gleichspringe weiter2
  zeige r
  zeigeweiter
  zeigeweiter
  zeigeweiter
  zeigeweiter
  zeigeweiter
  hole r
springe ganzoben

! Umgebung bekannt!

markiere weiter2
zeige r
macheschrittrueckwaerts
zeigeweiter
hole r
zeige südost
kopiere
springe oben

! Unterprogramm Umgebung "leer"?

markiere leertest
  zeige südost
  hole test
  kopiere
  markiere testoben1
    zeige test
    pruefefeld
    nullspringe leertestzurueck
    zeige test
    zeigeweiter
    hole test
    gleichspringe schluss
  springe testoben1

markiere schluss
```

Abb. 81: Irrwege im Homunkulus-Modell

Zu Beginn eines Durchlaufs zur Suche des Ausgangs wird überprüft, ob sich um die Figur herum noch Merkmale befinden (Unterprogramm 'leertest'). Ist das der Fall, bricht die Figur ihren Weg ab. Danach merkt sich die Figur die Umgebung ('erinnere') im Langzeitgedächtnis L. In der anschließenden Schleife ('markiere oben' bis 'springe oben') wird geprüft, welches Nachbarfeld noch frei und unbekannt ist. Sollte sich keines finden lassen, ist die Figur in einer Sackgasse und stellt ihren Lauf ein ('gleichspringe schluss'). Sonst macht sie einen Schritt in die durch die Schleife ermittelte Richtung auf ein freies Nachbarfeld und schaut sich um ('schaue'). Mit ihrem Umgebungseindruck wird die Matrix L abgefragt ('frageweiter') und sollte diese antworten, dass die Umgebung bereits bekannt sei, geht die Figur wieder einen Schritt zurück und versucht es in einer anderen Richtung erneut mit einem Schritt.[139] [140]

Die Abbildung 81 zeigt zwei typische Wege des Homunkulus durch den Merkmalswald. Links fand er einen Weg ins Freie und stoppte daher. Rechts endete sein Lauf, weil er an einen Ort kam, an dem er bereits gewesen war.

Damit sind Beispiele für künstliche Wesen gegeben, die sich in ihren Leistungen einerseits auf ihr Gedächtnis verlassen können und andererseits lernfähig zeigen. BRAITENBERG gibt als Wesen 7 eines an, welches dank einer Rolle Spezialdrahts der Marke „Mnemotrix“[141] in der Lage ist, sich auf eine größere

[139] Zur Kodierung des Umgebungseindrucks s. Kap. 8.

[140] Der Programmtext zwischen 'gleichspringe weiter2' und 'springe ganzoben' stammt von meinem Sohn Lukas Dierks, der mit dieser seiner Idee erreicht, dass der Homunkulus sich in Bewegungsrichtung anfänglich stets nach rechts orientiert und daher im Umlaufsinne des Richtungskreises keine mögliche Richtung auslässt.

[141] „Es stört mich nicht, wenn es die Elektroniker beim Lesen schaudert. Sie wissen genau, dass, selbst wenn Mnemotrix als Draht nicht kommerziell erhältlich ist, er durch eine einfache Schaltung simuliert werden kann.“ aus: [Braitenberg 86, S. 31]

Situationsvielfalt einzustellen. Er beobachtet dabei ein Wesen dieses Typs, welches einige (negative) Erfahrungen auf eine neue Situation übertrug. Im „Robot"- und „Homunkulus"-Modell trifft man ähnliche Wesen, in denen statt Mnemotrix Assoziativmatrizen das Speichern von Erfahrungen übernehmen.

Würde man eine VIDAS -Maschine auf Räder setzen und auf ihr ein solches Programm laufen lassen, entstünde so etwas wie ein Fahrzeug, das störunanfällig seinen Weg findet. Autos ohne menschliche Fahrer sind gelegentlich Thema in der Presse, wenn etwa sieben Pentium-Computer mit ausgefeilter Software einen Geländewagen selbsttätig mit Hilfe von Radargeräten, Laser-Entfernungsmessern, Videokameras und GPS-Navigationssystem steuern.[142] Am 8. Oktober 2005 wird auf einer Wüstenpiste im Südwesten der USA bei einem Rennen für solche fahrerlosen Vehikel um ein Preisgeld von 2 Millionen Dollar gegeneinander angetreten, was das Interesse an den Ergebnissen der Forschung auf diesem Gebiet unterstreicht. Schon im August 1989 versuchte man im ALVINN-Projekt[143] ein Fahrzeug mit Hilfe eines neuronalen Netzes einer gewundenen Straße folgen zu lassen. Damals schaffte man das mit einer Geschwindigkeit von 3,5 $\frac{miles}{h}$,[144] mit heutiger Technik gelingen gelegentlich Geschwindigkeiten von über 50 $\frac{km}{h}$. Der verantwortliche Ingenieur von VW für das „Stanley" getaufte Projekt, Carlo Rummel, nennt als Gründe für die begrenzte Geschwindigkeit die Störungen, die von Wasserlöchern, Schmetterlingen oder kugeligen Büschen ausgelöst werden.[145] Diese Problemlage lässt ahnen, dass ein weiterer Ausbau des hier vorgestellten „Robot"-Modells der VIDAS -Maschine in diesem Anwendungsfeld Nutzen brächte.

[142] vgl. Marco EVERS: „Stanleys Suche nach der Straße" in: „Der Spiegel" 24/2005 vom 13. Juni 2005

[143] ALVINN — Autonomous Land Vehicle in a Neural Network

[144] vgl. [Touretzky 89, S. 231]

[145] Carlo Rummel: „Es ist eine Sache der Software-Robustheit, wenn von unterschiedlichen Sensoren widersprüchliche Informationen kommen. Ein Teilproblem ist da zum Beispiel ein Wasserloch, das sich für den Sensor als glatte Oberfläche darstellt - so wie eine befahrbare Straße.", s. Spiegel Online, 24. Juni 2005

8 Kodierungen

In den Anwendungsbeispielen des Kapitels 7 speicherten die Figuren in den Modellen „Robot“ und „Homunkulus“ ihre Umgebungen im Langzeitgedächtnis L ab, um später darauf zurückgreifen zu können. Auf die Frage, wie eine solche Umgebung in eine Bitfolge umgesetzt wird, wurde dabei noch nicht eingegangen. Zwar hat PALM in [Palm 82] im Kapitel “How to build well-behaving machines“ gezeigt, dass und wie sich jede Eingabe für eine gedachte Maschine im Sinne seines Gehirnmodells durch eine Bitfolge darstellen lässt, doch hat man für das optimale Speichern von Bitfolgen in Assoziativmatrizen auf die Anzahl an Einsen zu achten, die in einer Bitfolge stehen.

PALM gibt in „The PAN System and the WINA Project“[146] einen Überblick zur spärlichen Kodierung. Die Anzahl der zu verteilenden Einsen pro Bitfolge spielt bei der Speicherausnutzung eine Rolle und nimmt Einfluss auf die Kosten der Hardware.[147] Sind nur jeweils wenige Einsen pro Matrixspalte aufzusummieren, können einfachere Digitalzähler mit weniger Stellen gewählt werden. Für Autoassoziationen gibt PALM am gleichen Ort als maximale Ausnutzung $\frac{ln(2)}{4}$ (17,33 %) an. Um bei einer Assoziativmatrix der Art aus Kapitel 2.2, oben, die maximale Speicherausnutzung von $ln(2)$ (69 %) zu erreichen, gilt für die optimale Spärlichkeit p für große Bitfolgelängen n:

$$p = \frac{ld(n)}{n}$$

Da in dem genannten „Homunkulus“-Modell eine Bitfolge aus etwa n = 1.000 Bits besteht, wäre es demzufolge anzustreben, die Merkmals-Umgebung, in der sich die Figuren dort bewegen, auf 10 Einsen abzubilden ($p \approx 0,01$). An diesem Beispiel sei Vorgehen und Bedeutung dieses spärlichen Kodierens für den Einsatz von Assoziativmatrizen und also auch für die VIDAS -Maschine erläutert.

In [Bentz 89] und in [Hagström 96] wurden genauere Untersuchungen zum spärlichen Kodieren für verschiedene Implementationsformen einer Assoziativmatrix vorgenommen. Dazu wurde für die jeweilige Implementationsform (hier zum Beispiel der Feld-Feld-Darstellung) der Ausnutzungsgrad als Quotient aus den Bits des Dargestellten und den Bits für die Darstellung definiert. Nach dem Satz von SHANNON-WIENER liefert

$$I_z = -nz(p \cdot ld(p) + q \cdot ld(q))$$

[146] vgl. [Palm 93, 2. Kap.]
[147] Zum PAN-System s. Kap. 3.2

die Gesamtinformation (in Bit) in z binären Vektoren mit p als Spärlichkeit der „1“, $q = 1 - p$ und n als Anzahl der Matrixspalten. Damit ist der Zählerterm für den Ausnutzungsgrad gegeben; der Nennerterm ergibt sich als Summe von $n\alpha$ (Aufwand für die Verwaltung der Implementation mit α als Maschinenkonstante) und $npz \cdot ld(z)$ als Anzahl der Bits für die Darstellung der binären Vektoren.

Als Ausnutzungsgrad H erhält man somit

$$H(n,p,z) = -\frac{nz(p \cdot ld(p) + q \cdot ld(q))}{n\alpha + npz \cdot ld(z)}$$

und mit $\widetilde{\alpha} = \alpha \cdot \ln 2$

$$H(n,p,z) = -\frac{z(p \cdot \ln(p) + q \cdot \ln(q))}{\widetilde{\alpha} + pz \ln(z)}$$

Für eine Matrixgröße wie im „Homunkulus“-Modell ($n = z \approx 1.000$) sieht der Graph von H dann wie in Abb. 82 aus. Man erkennt, dass für Spärlichkeiten p, die größer als ein gewisses Maximum von H sind, die Wahl einigen Spielraum lässt, da die Steigung des Graphen zwar negativ aber nicht sehr groß ist.

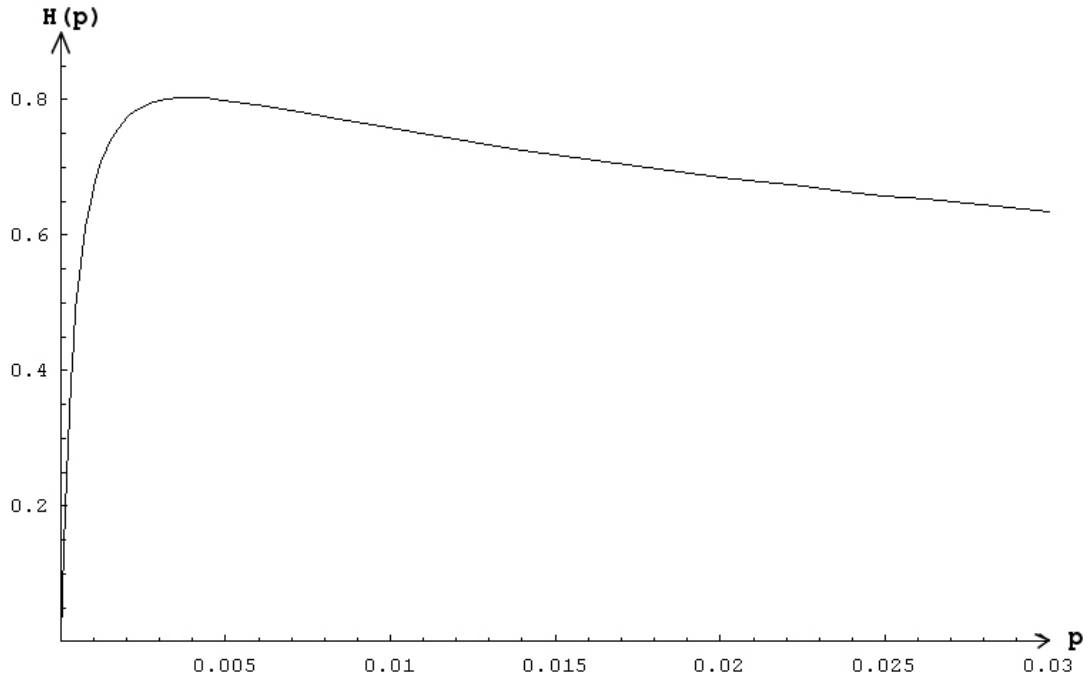


Abb. 82: Ausnutzungsgrad H in Abhängigkeit von der Spärlichkeit p

Da es um optimale Spärlichkeit p geht, wird $H(n,p,z)$ nach p abgeleitet und dann gleich Null gesetzt:

$$\frac{\partial H(n,p,z)}{\partial p} = -z\frac{(\ln(p)-\ln(q))(\widetilde{\alpha}+pz\ln(z))-(p\ln(p)+q\ln(q))z\ln(z)}{(\widetilde{\alpha}+pz\cdot\ln(z))^2}$$

und

$$0 = \widetilde{\alpha}\ln(p)-\widetilde{\alpha}\ln(q)-z\ln(q)\ln(z) \;\Leftrightarrow\; \frac{\ln(p)}{\ln(q)} = \frac{z\ln(z)+\widetilde{\alpha}}{\widetilde{\alpha}}.$$

Da für die Konstante $\widetilde{\alpha}$ die Werte in der Größenordnung von etwa 20 bis 40 liegen, gilt

$$\frac{z\ln(z)+\widetilde{\alpha}}{\widetilde{\alpha}} \gg 1$$

und also

$$\frac{\ln(p)}{\ln(q)} \gg 1,$$

woraus gefolgert wird, dass die Spärlichkeit p wesentlich kleiner als q sein muss $(q = 1-p)$. So ergibt sich mit $\ln(q) \approx -p$ und $z\ln(z) \gg \widetilde{\alpha}$ als Abschätzung die Optimalitätsbedingung:

$$-\frac{\ln(p)}{p} \approx z\frac{\ln(z)}{\widetilde{\alpha}}.$$

Setzt man in diese Gleichung für das „Homunkulus“-Modell $z = 1.000$ und $\widetilde{\alpha} = 20$ ein, liefert die Rechnung für die Spärlichkeit $p \approx 0,0127$.[148]

Wie sich die Anforderungen an die Spärlichkeit ändern, wenn die Matrizen größer werden, kann veranschaulicht werden (s. Abb. 83), indem der Graph von H hinsichtlich zweier Veränderlicher (Spärlichkeit und Größe) dargestellt wird. Der in Abb. 82 gezeigte Graph befindet sich in Abb. 83 im Bildhintergrund (z = 1.000). Für kleiner werdende Matrizen (für $z < 100$) ändert sich für die Wahl von p die Größenordnung erkennbar.

HAGSTRÖM untersucht in [Hagström 96] dann drei Kodierungen für Zeichenketten hinsichtlich ihrer Spärlichkeit. Er wählt zum Vergleich eine Buchstaben-Position-Kodierung, eine Doppelzeichen- und eine Tripelzeichenkodierung. Bei der Buchstaben-Position-Kodierung wird beispielsweise das ASCII-Alphabet zur Hilfe genommen, um jedem Buchstaben eine Ordinalzahl n zuzuordnen. Beim Kodieren der Zeichenkette in eine Bitfolge erhält jedes Zeichen einen bestimmten Abschnitt der Bitfolge und setzt in diesem dann das n. Bit auf „1“. Als Spärlichkeit errechnet HAGSTRÖM dafür $p = 0,025$.[149] Zur Doppelzeichenkodierung,

[148] alle Rechnungen für die Feld-Feld-Darstellung einer Assoziativmatrix

[149] s. [Hagström 96, S. 79]

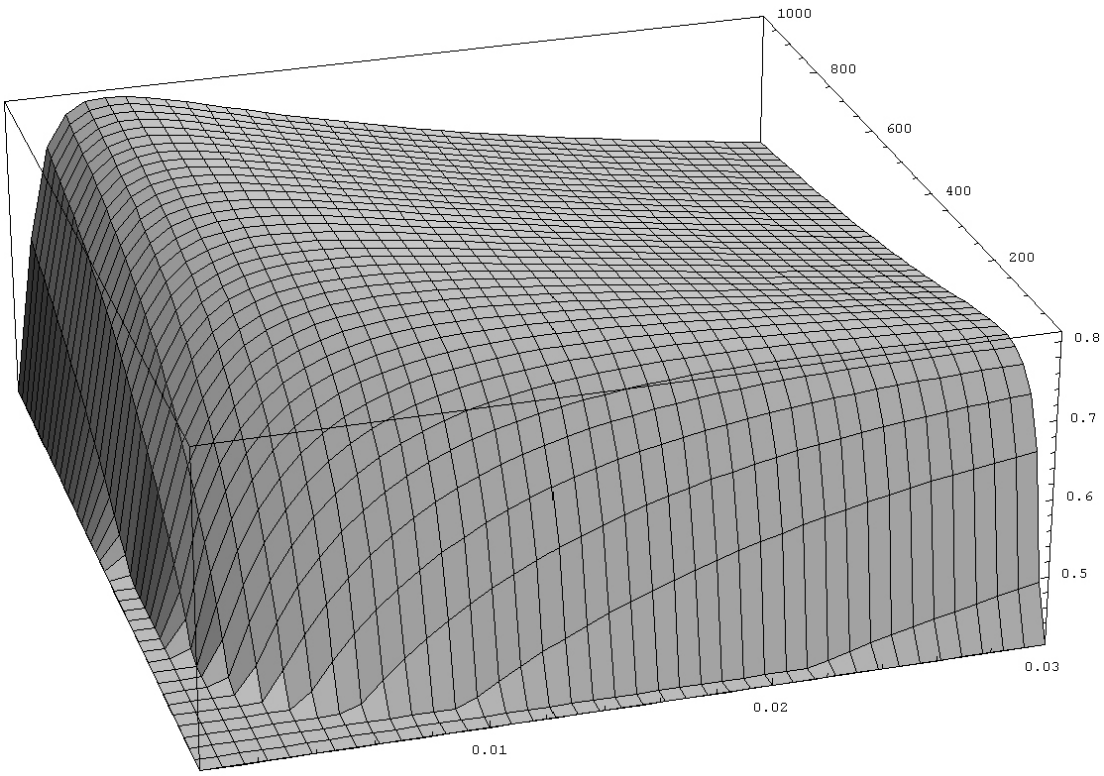


Abb. 83: Ausnutzungsgrad und Matrixgröße

bei der je zwei aufeinanderfolgende Zeichen der Zeichenkette eine bestimmte „1“ in der Bitfolge setzen, erhält er $p = 0,006875$. Und bei der Tripelzeichenkodierung, durch die je drei aufeinanderfolgende Zeichen eine „1“ setzen, berechnet er $p = 0,0001563$.

Für die Kodierung einer Umgebung im „Homunkulus“-Modell wurde eine Kodierung der Art Doppelzeichenkodierung gewählt, nur dass keine Zeichen sondern Merkmalswerte die Orte der zu setzenden Einsen in den zu lernenden Bitfolgen bestimmen. Kreisförmig um den Homunkulus herum werden dazu die Merkmalswerte ermittelt. Es wird zwischen 30 Merkmalen mit den Werten 1, 2, ..., 30 unterschieden. Diese Anzahl heiße M. Je zwei aufeinanderfolge Merkmalswerte m_1, m_2 setzen dann das $((m_1 - 1) \cdot M + m_2)$. Bit in der Bitfolge, die in die Matrix L eingetragen wird. Um optimale Spärlichkeit zu erreichen (hier also etwa 10 Bit), wurden die Beobachtungskreise um den Homunkulus entsprechend groß gewählt. Da in acht Himmelsrichtungen unterschieden wird, erhält man in nächster Umgebung jeweils 8 Bit. Das bei HAGSTRÖM erwähnte Anfangs- und

Schlusszeichen '#'[150] entfällt wegen der kreisförmigen Anordnung. Um auf die gewünschten 10 Bit aufzufüllen, wurden aus dem nächstgrößeren Umgebungskreis weitere Merkmalswerte hinzugenommen. Für die Doppelzeichenkodierung spricht, dass sich beim Ändern eines Merkmals nur zwei Bits in der Kodierung ändern (ähnlichkeitserhaltend). In dieser Hinsicht kommt für das genannte Modell auch die Positionskodierung in Betracht, deren Spärlichkeit allerdings größer wäre. Die Praxis im Umgang mit Assoziativmatrizen lehrt,[151] dass der geeigneten Kodierung im jeweiligen Anwendungsfall eine wichtige Rolle zukommt. Auf die von KOHONEN diskutierten Ähnlichkeitsabstände ist dabei das Augenmerk zu richten.[152]

[150] s. [Hagström 96, S. 77]
[151] vgl. [Dierks 88b], Kodierung von Postleitzahlen der USA
[152] vgl. [Kohonen 88, Kap. 2.2]

9 Zusammenfassung und Ausblick

Als Konrad ZUSE 1938 seinen ersten Rechner im elterlichen Wohnzimmer in Betrieb nahm, war damit seine Idee einer Maschine zur programmgesteuerten Verarbeitung von Zahlen in die Tat umgesetzt worden, auch wenn diese aufgrund ihres mechanischen Aufbaus[153] noch unzuverlässig arbeitete.[154] Das Blockschaltbild in Abb. 84 stellt das Rechenwerk zu Recht in den Mittelpunkt der Z1.[155]

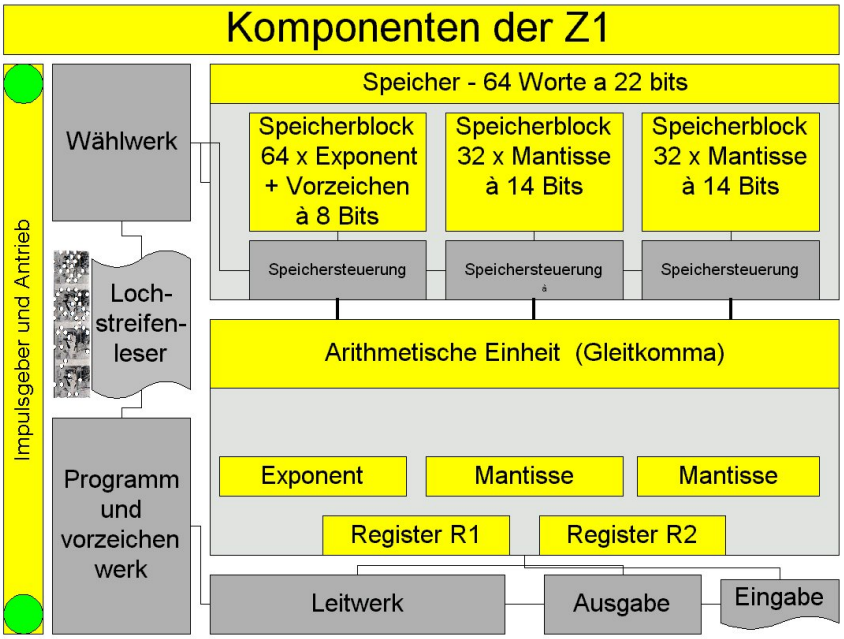


Abb. 84: Blockschaltbild der Z1

Die Aufgabe, eine frei programmierbare Maschine zu konstruieren, die erstens nicht das Speichern und Verarbeiten von Zahlen als Hauptziel umsetzt und zweitens auch noch robust, störunanfällig arbeitet, stellte sich damals nicht, obwohl ZUSE beim Zusägen der Bleche für die Z1 eine systemimmanente Fehlertoleranz sicherlich als wünschenswert durch den Kopf gegangen sein mag.[156]

Ein Programm für die Z1 wurde ähnlich wie beim Jacquard-Webstuhl auf einem Lochstreifen gespeichert, befand sich also außerhalb des eigentlichen Rechners.

Die in dieser Schrift vorgestellte VIDAS -Maschine setzt Assoziativmatrizen in ihren Mittelpunkt, die Bitfolgen einander zuordnen, unabhängig davon, was diese

[153] Konrad Zuse stellte die nötigen Bleche mit Hilfe einer Laubsäge her.
[154] s. TU Berlin, http://irb.cs.tu-berlin.de/~zuse/Konrad_Zuse/de/
[155] Abb. von http://irb.cs.tu-berlin.de/~zuse/Konrad_Zuse/img/z1-bl.jpg
[156] Die Z1 besteht aus etwa 20.000 Einzelteilen.

Bitfolgen repräsentieren mögen. Nicht nur die zu verarbeitenden Daten, auch die zugehörigen Programme werden in den Matrizen abgelegt.[157] John VON NEUMANN beschrieb dieses Prinzip 1945. Das Anwendungsprogramm und seine Daten in einem Speicher unterzubringen, das erspart nicht nur den Transport von Daten zwischen Matrix-Speicher und Wirtsrechner wie im PAN-System,[158] sondern ermöglicht auch in einfacher Weise bedingte und unbedingte Sprünge im Programmablauf.[159]

Assoziativmatrizen statt eines Rechenwerks als Kern eines Prozessors zu wählen, schafft einen allgemeineren Ansatz von dem, was heute „Rechner“ oder „Computer“ genannt wird. „L'ordinateur“, das französische Wort für diese Art Maschinen, lässt anklingen, dass es um die Anordnung von (abzählbaren) Dingen geht. In diesem Sinne mag man die VIDAS -Maschine als eine verstehen, die Objekte anreiht, in Assoziationsketten verbindet und dem Beobachter beim Blick in ihre Matrizen dennoch den Eindruck von Unordnung vermittelt. Sie lässt nicht erkennen, was in ihren Matrizen gespeichert wurde und in welcher Reihenfolge das geschah. Demonstriert man zudem die Störunanfälligkeit der Maschine, indem man in der vermeintlichen „Unordnung“ der Matrix Bits beliebig setzt oder löscht, wird der Beobachter im System erst recht keinerlei Ordnung vermuten. Das macht die Maschine in kryptologischer Hinsicht, aber auch mit Blick auf die Angreifbarkeit durch Viren interessant, da diese den Programmcode, in den sie sich einfädeln wollen, nicht finden.

Die architektonischen Unterschiede zu einem Rechner klassischen Zuschnitts sind nicht nur von Interesse für die Theorie des Rechnerbaus, sie tragen zugleich in sich eine Vorstellung davon, dass sich auch im Gehirn Bereiche unterscheiden lassen, denen verschiedene Aufgaben zukommen (s. Abb. 85).

Es war nicht die Absicht des Projekts, Gehirnstruktur nachzubilden. Doch während der Entwicklung zeigte sich, dass die Abarbeitung von Programmen durch eine Gliederung der VIDAS -Maschine in sechs Matrizen ermöglicht wird. Auch wenn die vier programmspeichernden Matrizen $A, B, P1, P2$ beim Bau physikalisch zu einer großen Matrix zusammengesetzt werden, so ändert sich dadurch an ihren getrennten Aufgabenbereichen nichts. Was entstünde wohl, wenn man in Umkehrung dieses Vorgehens, Assoziativmatrizen nach dem biologischen Vorbild zu einer Maschine zusammenfügte und womöglich die Matrizen für das Kurz- und Langzeitgedächtnis K und L durch Matrizen für das Sehen, das Hören, die

[157] ebenfalls als Bitfolgen

[158] s. Kap. 3.2

[159] Der Lochstreifenleser der Z1 wurde mit einer Taktfrequenz von 1 Hz durch einen Elektromotor vorangetrieben. Ein bedingter Rücksprung im Programmablauf ließ sich da schwerlich verwirklichen.

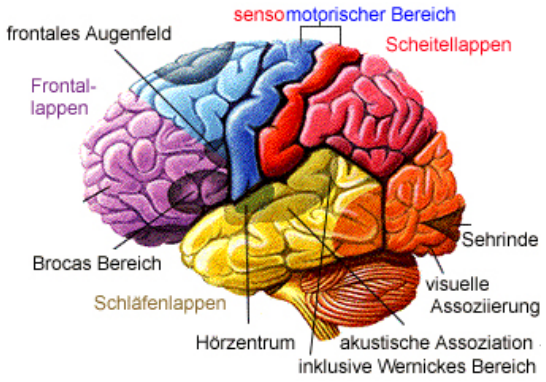


Abb. 85: Aufgabenfelder im Gehirn

Sensomotorik ersetzte? An dieser Stelle öffnet sich ein weiteres, spannendes Betätigungsfeld.

Es wurde im Kapitel 4.3 gezeigt, dass das Lernen durch die VIDAS -Maschine in einem Taktschritt erfolgen kann, das Abfragen jedoch linear von der Anzahl der Zeilen der Assoziativmatrizen abhängt. Das liegt an der eingesetzten digitalen Zählertechnik. Um einen Weg zu finden, auch das Abfragen in einem Taktschritt zu erledigen, könnte man Analogtechnik einsetzen (Operationsverstärker). Deren Ungenauigkeit ließe sich womöglich durch die systemimmanente Robustheit ausgleichen. Am Ende einer solchen Entwicklung stünde eine Maschine, die unter Berücksichtigung der technischen Eigenheiten der Analogtechnik etwa 10^8 Lern- und Abfragevorgänge pro Sekunde bewältigte, egal wie groß die eingebauten Matrizen sind. Die sich so ergebende Taktfrequenz von 100 MHz lässt sich mit der zehnmal höheren Taktfrequenz eines herkömmlichen Rechners nicht vergleichen, da dieser für die Simulation eines einzigen Lern- und Abfragevorgangs schon Tausende von Taktzyklen benötigte.

So liefert diese Arbeit zum Schluss einen Ausblick auf eine VIDAS -Maschine, die störunanfällig in hoher Geschwindigkeit nicht nur lernen kann, sondern ihre Daten in ebenfalls hoher Geschwidigkeit dank des in ihr wirkenden Programms fehlertolerant verarbeitet. Und selbst eine in digitaler Technik aufgebaute Maschine, deren Abfragegeschwindigkeit bei einer Millisekunde liegen dürfte, verspricht durch ihre Eigenschaften einen Nutzen in zahlreichen Anwendungsfeldern, welche vom Ungefähren geprägt sind. Der Schritt von den in dieser Arbeit gezeigten Schaltungen zur tatsächlichen Hardware ist nicht groß, da der Plan bis herunter zu jedem einzelnen Flip-Flops fertig vorliegt. Jetzt könnte die VIDAS -Maschine mit Hilfe eines Programms zur Eingabe der Schaltung für einen anschließenden

ASIC-Fertigungsprozess[160] fertiggestellt werden. Erste Experimente mit der Maschine sind in absehbarer Zeit zu erwarten. Im Unterschied zur Z1 müssen dafür zum Glück nicht mehr Tausende von Blechen zugesägt werden. Das Anlöten einiger Hundert Beine genügt.

[160] ASIC — Application Specific Integrated Circuit, d.i. Anwendungsspezifische integrierte Schaltung

10 Verzeichnisse

Abbildungsverzeichnis

Literatur

[Abelson 83] Harald Abelson: „Einführung in LOGO“, übersetzt und bearbeitet von Herbert Löthe, IWT Verlag, Vaterstetten bei München 1983, ISBN 3-88322-023-X

[Bentz 88] Hans Joachim Bentz: „Ein Gehirn für den PC - Schneller Matrixspeicher als assoziatives Gedächtnis“, in: „c't Magazin für Computertechnik“, Heft 10/1988, Verlag Heinz Heise, Hannover 1988

[Bentz 89] Hans-J. Bentz, Michael Hagström, Günther Palm: „Information Storage and Effective Data Retrieval in Sparse Matrices“, in: „Neural Networks“, Vol. 2, S. 289-293, Maxwell Pergamon Macmillan, 1989

[Braitenberg 86] Valentin Braitenberg: „Künstliche Wesen - Verhalten kybernetischer Vehikel“, Vieweg, Braunschweig 1986

[Dierks 88a] Andreas Dierks: „$ASSO^3$ - Entwicklungs- und Interpreterschalen für ein Assoziativspeicherkonzept mit Matrizen“, Seminarunterlage, Universität Osnabrück, 1988

[Dierks 88b] Andreas Dierks: „ZIP-Extra — Testumgebung für assoziative ZIP-Matrizen“, Universität Osnabrück, 1988

[Dierks 89a] Andreas Dierks: „Logikschaltpläne für Assoziativmatrizen“, Seminarunterlage, Universität Osnabrück, 1989

[Dierks 89b] Andreas Dierks: „Erkennen von Keywords in realen Leserdaten mit Hilfe spärlich codierter Assoziativmatrizen“, Universität Osnabrück, 1989

[Fields 04] R. Douglas Fields: „Die unbekannte Seite des Gehirns“, in: „Spektrum der Wissenschaft“ Nr. 9/2004, S. 46-56, Spektrum der Wissenschaft Verlag, Heidelberg 2004

[Grams 86] Timm Grams: „Codierungsverfahren“, Bibliographisches Institut, Mannheim 1986, ISBN 3-411-00625-0

[Hagström 96] Michael Hagström: „Textrecherche in großen Datenmengen auf der Basis spärlich codierter Assoziativmatrizen“, Dissertationsschrift, Universität Hildesheim, 1996

[Hamming 87] Richard Wesley Hamming: „Information und Codierung“, VCH Verlag, Weinheim New York 1987, ISBN 3-527-26611-9

[Hecht-Nielsen 91] Robert Hecht-Nielsen: „Neurocomputing“, Addison-Wesley Publishing, Reading 1991, ISBN 0-201-09355-3

[Heitland 94] Michael Heitland: „Einsatz der SpaCAM-Technik für ausgewählte Grundaufgaben der Informatik“, Dissertationsschrift, Universität Hildesheim, 1994

[Hoffmann 92] Norbert Hoffmann: „Simulation Neuronaler Netze — Grundlagen, Modelle, Programme“, 2., verbesserte Auflage, Vieweg, Braunschweig/Wiesbaden 1992, ISBN 3-528-15140-4

[Holthausen 94] Olaf Holthausen: „Ein Vergleich verschiedener Implementationen binärer neuronaler Assoziativspeicher“, Dissertationsschrift, Universität Ulm, 1994

[Jütting 90] Andreas Jütting: „Auslesegüte defekter Assoziativspeicher“, Diplomarbeit, Universität Osnabrück, 1990

[Kohonen 88] Teuvo Kohonen: „Self-Organization and Associative Memory“, Second Edition, Springer-Verlag Berlin Heidelberg New York, o.J., ISBN 3-540-18314-0

[Krätschmar 88] Roland Krätschmar: „Assoziative Speicherung auf neuronaler Basis“, Rheinische Friedrich-Wilhelms-Universität Bonn, 1988

[Lindenmair95] Wolfgang Lindenmair: „Arbeitshefte Informatik — Neuronale Netze“, Klett, Stuttgart 1995

[Palm 80] Günther Palm: „On Associative Memory“, in: „Biological Cybernetics“ Nr. 36, S. 19-31, Springer-Verlag, 1980

[Palm 82] Günther Palm: „Neural Assemblies - An Alternative Approach to Artificial Intelligence“, Springer-Verlag, Berlin Heidelberg New York 1982, ISBN 3-540-11366-5

[Palm84] Günter Palm: „Parallel Processing for Associative and Neural Networks“, in: „Biological Cybernetics“ Nr. 51, S. 201-204, Springer-Verlag, 1984

[Palm 88] Günther Palm: „Assoziatives Gedächtnis und Gehirntheorie“, in: „Spektrum der Wissenschaft“ Nr. 6/1988, S. 54-64, Spektrum der Wissenschaft Verlag, Heidelberg 1988

[Palm 90] Günther Palm: „Cell Assemblies as a Guideline for Brain Research“, in: „Concepts in Neuroscience“ Vol. 1 No. 1, S. 133-147, World Scientific Publishing, 1990

[Palm 93] Günther Palm: „The PAN System and the WINA Project“, in: „Euro-ARCH ’93. Europäischer Informatik Kongreß Architektur von Rechensystemen“ (Spies, P.P, Hrsg.), Springer-Verlag, Berlin Heidelberg 1993

[Riemschneider 96] Karl-Ragmar Riemschneider: „Parallele Hardware für Backpropagation-Netze auf der Basis stochastischer Rechenwerke“, Dissertationsschrift, Universität der Bundeswehr Hamburg, 1996

[Tietze/Schenk 02] U. Tietze, Ch. Schenk: „Halbleiter-Schaltungstechnik“, 12. Auflage, Springer Verlag, Berlin Heidelberg New York 2002, ISBN 3-540-42849-6

[Touretzky 89] David S. Touretzky, Dean A. Pomerleau: „What’s Hidden in the Hidden Layers?“, in: „BYTE“ Vol. 14 No. 8, McGraw-Hill, 1989

[Ulbrich91] Holger Ulbrich: „Demonstrationsexperimente an Neuronalen Netzen“, Diplomarbeit, Fachhochschule Darmstadt, 1991

Index

11 Anhang

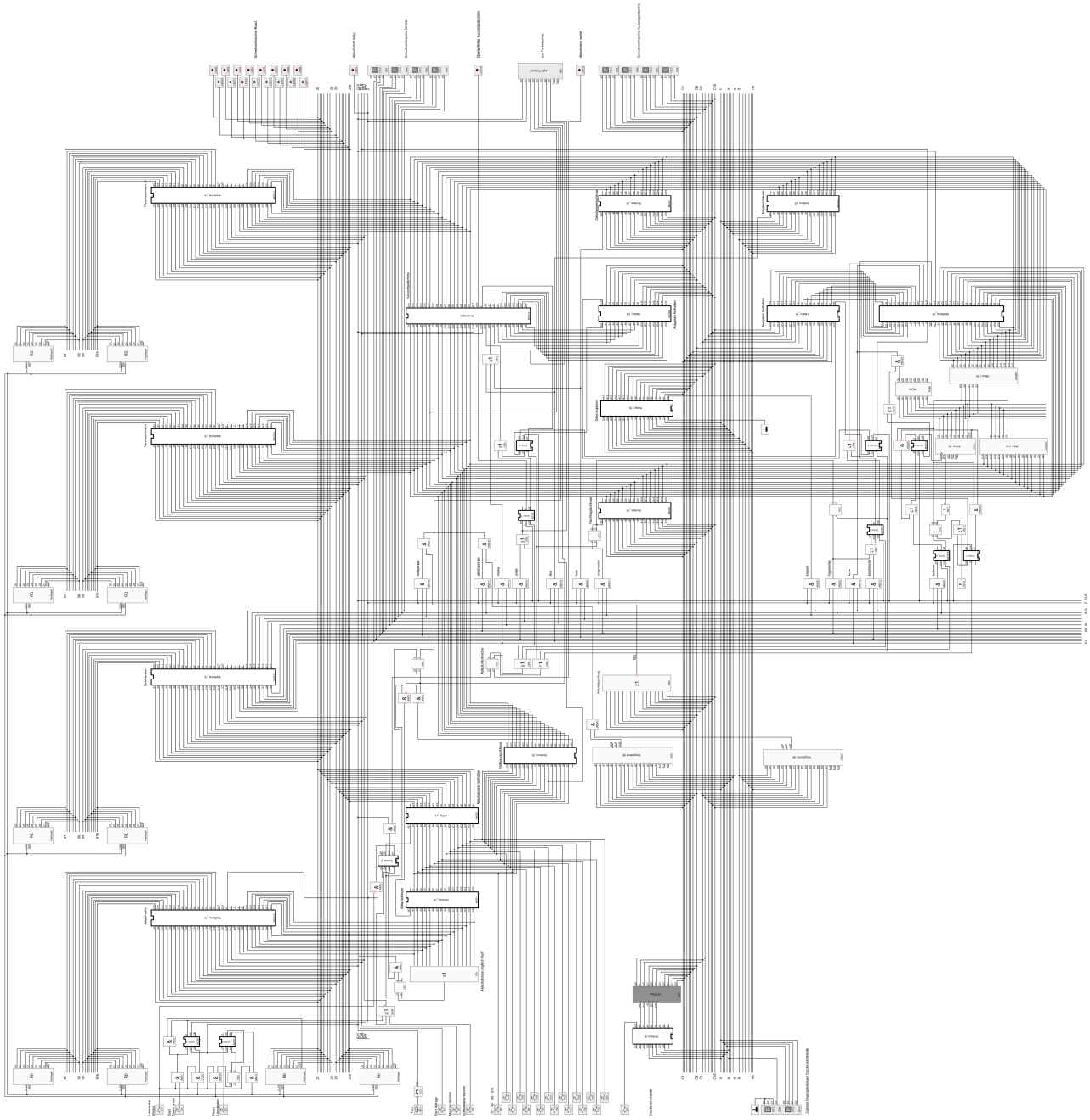


Abb. 86: VidAs 9

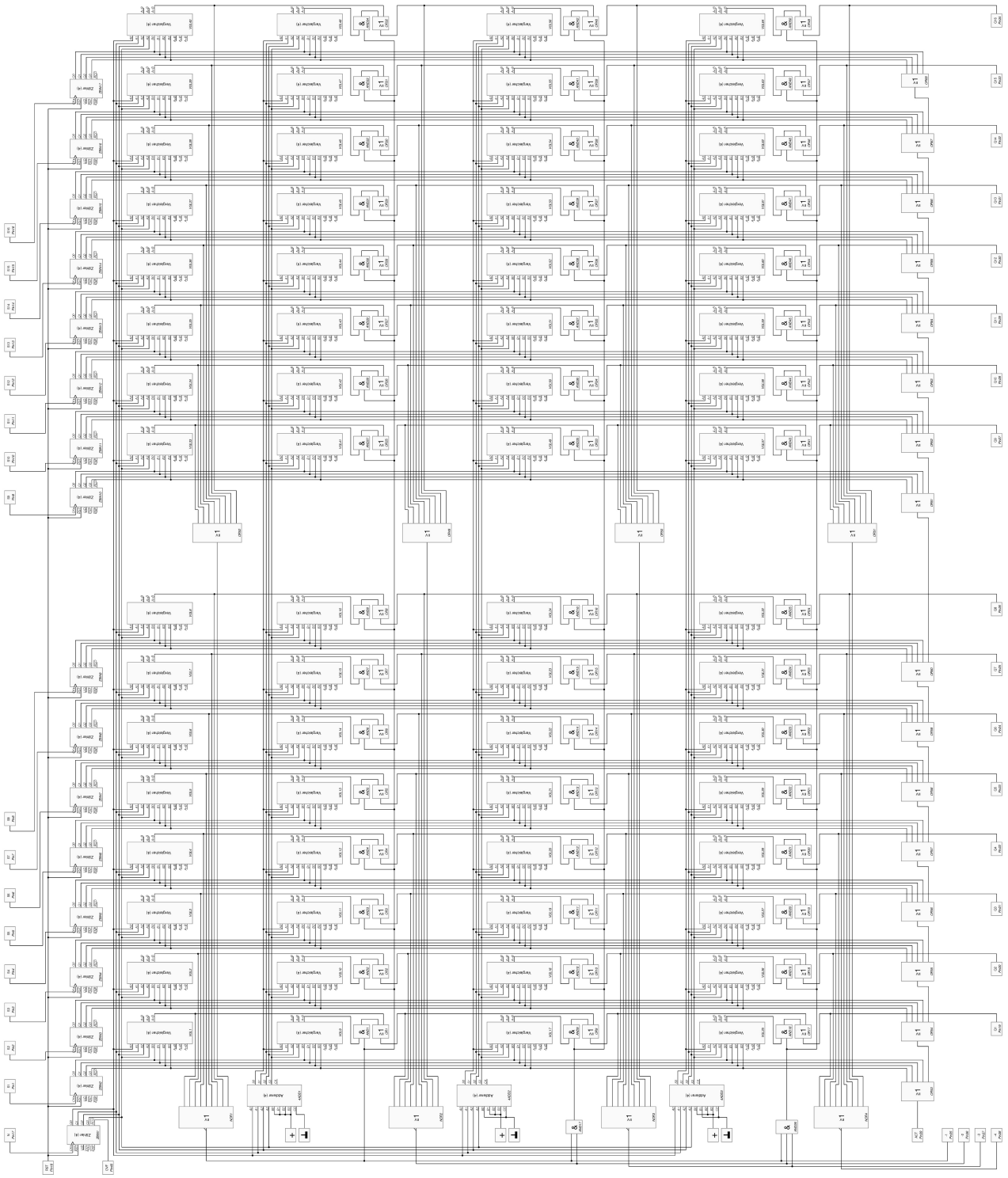


Abb. 87: Schwellwertlogik (Makro Schwell_16)

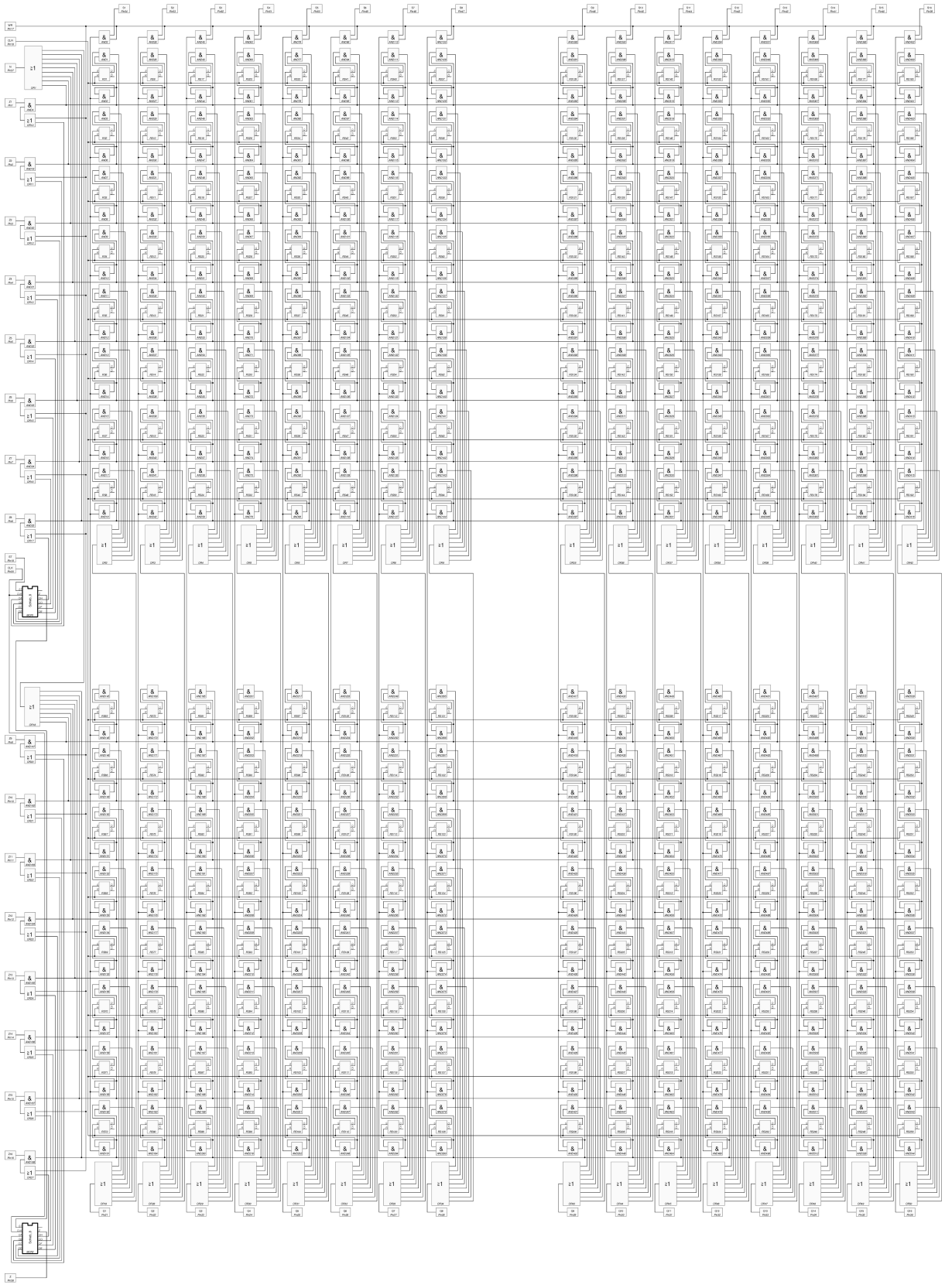


Abb. 88: Assoziativmatrix 16 x 16 (Makro Matrix_16x16)

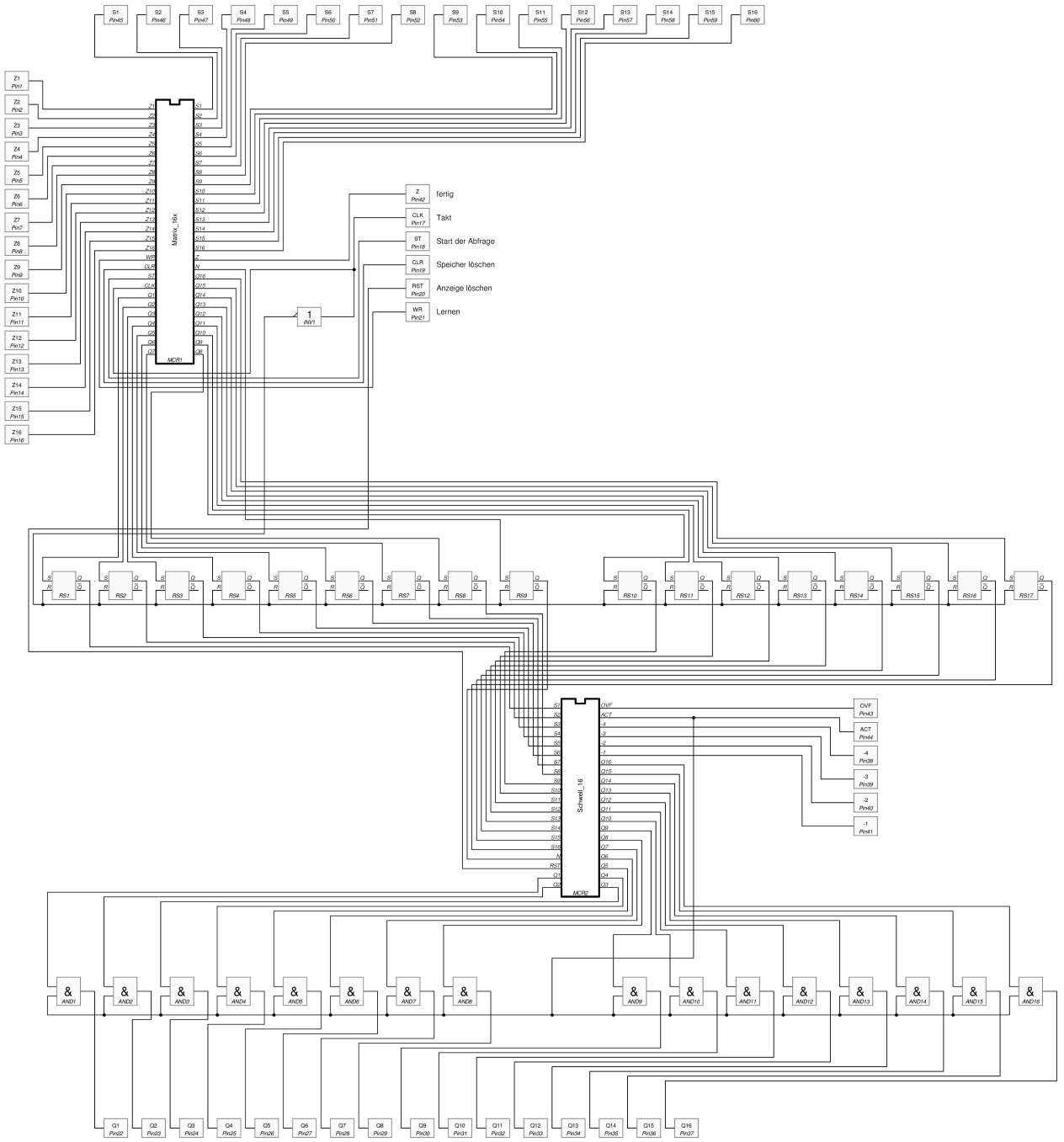


Abb. 89: Matrix mit Schwellwertlogik (Makro MatSchw_16)

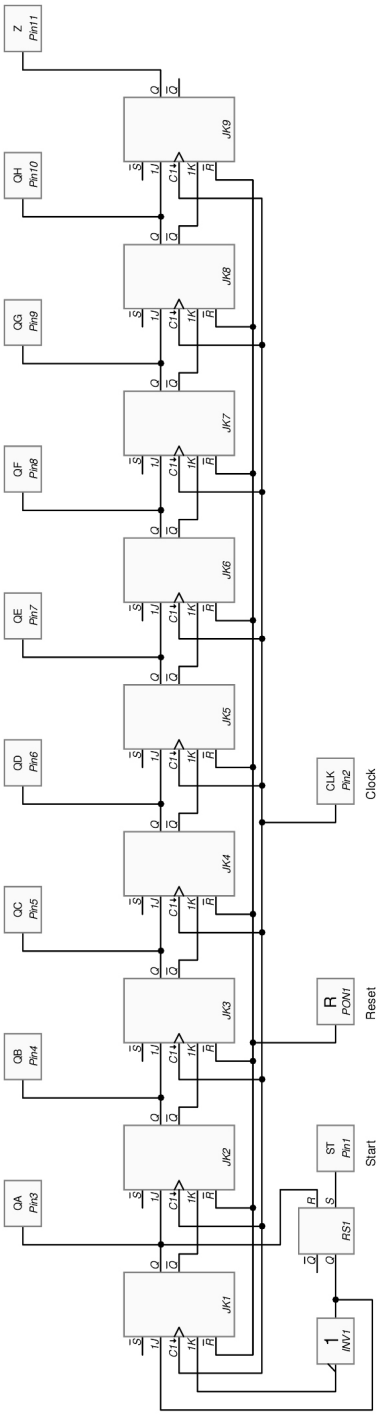


Abb. 90: Schieberegister zum Abfragen der Matrixzeilen (Makro Schieb_8)

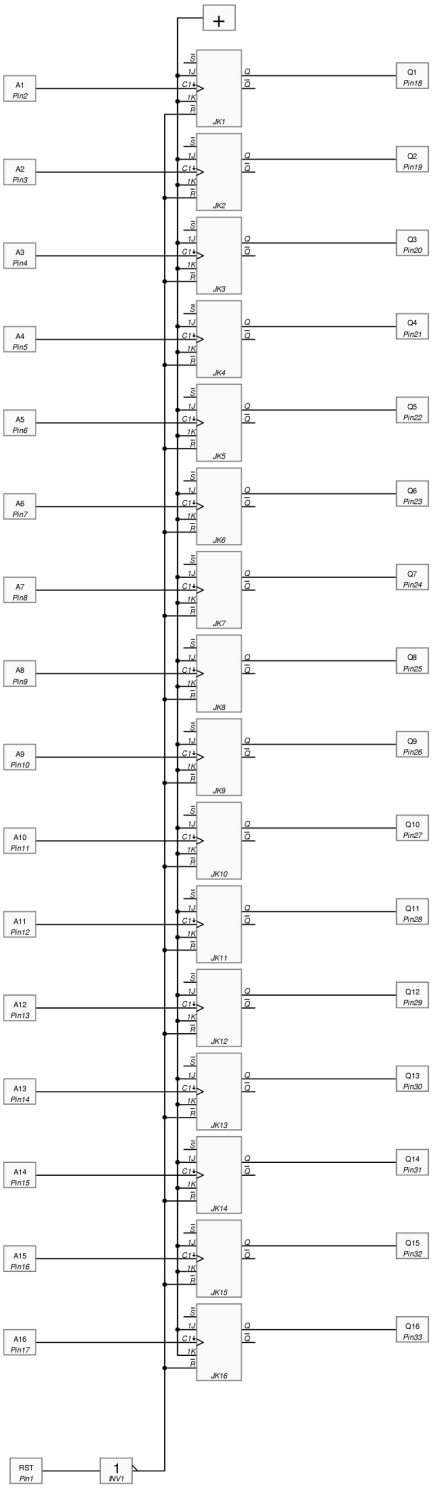


Abb. 91: JK-Flip-Flops zum Festhalten der Ablaufadresse (Makro JKFlip_16)

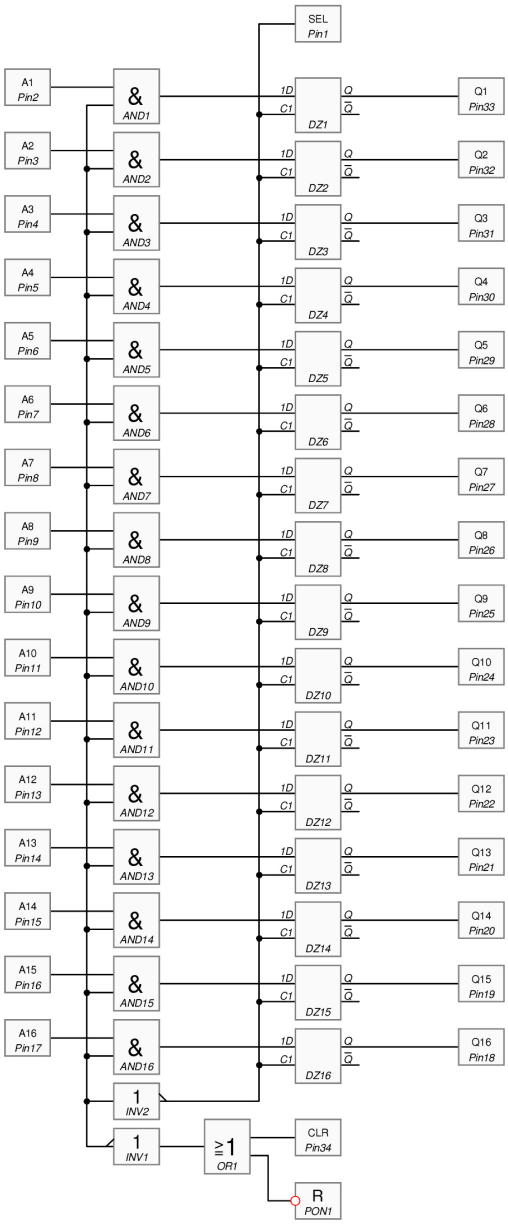


Abb. 92: RS-Flip-Flops zum Halten der Matrixausgaben (Makro Halten_16)

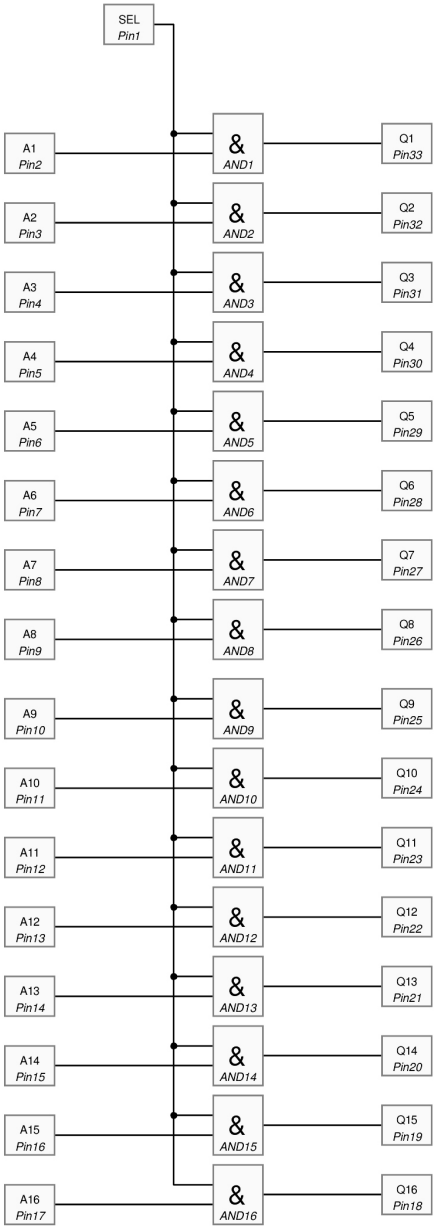


Abb. 93: Reihe von UND-Gattern zum programmgesteuerten Durchreichen von Daten (Makro Schleus_16)

Lebenslauf

25.10.1954	Geburt in Hamburg
1961 - 1965	Besuch der Volksschule Langenfort in Hamburg-Barmbek
1965 - 1970	Besuch des Gymnasiums St. Georg für Jungen in Hamburg
1970 - 1974	Besuch des Gymnasiums Schwarzenbek in Schwarzenbek
1974 - 1978	Dienst in der Bundesluftwaffe
1978 - 1984	Studium für das Lehramt an Gymnasien im Lande Niedersachsen mit den Fächern Mathematik und Physik an der Universität Osnabrück
1980 - 1984	Anstellung als wissenschaftliche Hilfskraft im Rechenzentrum der Universität Osnabrück mit den Aufgabenbereichen Benutzerberatung, Übungsbetreuung und Softwarewartung
1984 - 1986	Referendariat am Studienseminar Osnabrück
1984 - 1988	Studium der Informatik an der Universität Osnabrück
1986 - 1989	Anstellung an einer Schule für EDV-Berufe in Osnabrück als Lehrer für Mathematik, Software-Engineering, Algorithmen und Datenstrukturen, Programmiersprachen, hauptsächlich in Maßnahmen zur Mathematisch-Technischen Assistenz (MTA) und zum DV-Kaufmann; dort ab Anfang 1987 EDV-Fachbereichsleiter und Leiter des Hardware-Labors
1987 - 1990	Mitarbeit in der Arbeitsgruppe zur Neuroinformatik bei Prof. Dr. Bentz und Prof. Dr. Dr. Zielke, Osnabrück
1988	Berufung in den Prüfungsausschuss für MTA der Industrie- und Handelskammer Osnabrück-Emsland
1989 - 1990	Anstellung als Entwicklungsingenieur der Firma AEG Electrocom GmbH, Konstanz, an der Universität Osnabrück, Fachbereich Mathematik/Informatik; Erforschung der Einsatzmöglichkeiten von Assoziativmatrizen zur Mustererkennung bei Briefsortiermaschinen
1989 - 1990	Wahrnehmung von Lehraufträgen an der Fachhochschule Osnabrück, Fachbereich Werkstofftechnik, im Lehrgebiet Angewandte Mathematik und EDV
1990 - 2002	Anstellung als Lehrer für Mathematik, Physik und Informatik an der Liebfrauenschule Vechta, Unterricht in den Klassen 5 bis 13, Sammlungsleitung Physik
1992 - 1993	Anstellung an der Universität Hildesheim als wissenschaftlicher Mitarbeiter im Rahmen einer Nebentätigkeit im Studiengang Informatik
ab 2002	Anstellung an der Universität Hildesheim als wissenschaftlicher Mitarbeiter im Institut für Mathematik und Angewandte Informatik, Lehrtätigkeit in Mathematik und Informatik, Forschung im Bereich Assoziativmatrizen und störunanfällige Maschinen